EDT V17.0A Unicode Mode

Statements

Edition December 2007

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

Contents

1 Preface

EDT is the BS2000 file editor, used for the user-friendly creation and editing of BS2000 files in SAM and ISAM formats, as well as text-type library elements and POSIX files.

The repetitive operations which occur during editing such as deleting, modifying, inserting and copying records and characters, searching for records containing certain character strings, outputting records etc. are performed using powerful yet easy-to-learn statements.

EDT V17.0A can be used in an extended Unicode mode and a V16.6-compatible compatibility mode.

- In Unicode mode, EDT V17.0A can edit character sets coded in Unicode and other character sets. Users benefit from easy-to-handle support capabilities such as, for example, the ability to edit differently coded files in various EDT work files simultaneously. In addition, there is no longer any restriction to line length (previously 256 characters). When reading from and writing to files, EDT is able to process all the record lengths provided by DMS and LMS. In the case of POSIX files, it is able to process files with a maximum length of 32768 characters.

  The fact that a Unicode representation is used internally in EDT means that the compatibility of all the various interfaces at which users previously had direct access to the internal EDT data cannot be maintained. This applies to the old L mode subroutine interface, the former @RUN interface and the Locate mode of the IEDTGLE interface. These interfaces can therefore no longer be used in Unicode mode.

- The compatibility mode provides the full functionality of EDT V16.6B with only slight extensions.

Even though EDT has been designed to be an interactive program, it is also able to process files and library elements in batch mode.

File editing operations which have to be performed frequently in identical or similar forms can be programmed using EDT procedures.

EDT can call other programs as subroutines and can itself be called as a subroutine by a user program.

1.1 Structure of the EDT documentation

The manuals

- EDT V17.0A Unicode Mode Statements
- EDT V17.0A Unicode Mode Subroutine Interface

describe EDT's Unicode mode. The manuals

- EDT V16.6B Statements
- EDT V16.6A Subroutine Interface

contain a description of the compatibility mode.

In addition, the EDT V17.0A Statements manual contains a section describing the extensions in the compatibility mode compared to EDT V16.6B.

The manuals dealing with the EDT statements describe the fundamental concepts of EDT in each of these modes and can be used as a reference for the many EDT statements.

The manuals dealing with the subroutine interface describe how the user can write programs which can be called by EDT or which themselves call EDT as a subroutine can be programmed. These manuals can only be used properly in combination with the manuals describing the EDT statements.

1.2 Target groups for the EDT manuals

While the manual dealing with the EDT statements is intended for EDT novices and users, the manual dealing with the EDT subroutine interfaces is intended for experienced EDT users and programmers who want to employ EDT in their own programs.

This manual, which deals with the EDT statements, is intended for users who are not yet familiar with EDT through to experienced EDT users for whom chapter 9 “EDT statements” which contains a description of all the EDT statements will constitute a valuable reference document. The section on “EDT procedures” will be of great help to EDT users who want to write their own EDT procedures or modify existing EDT procedures.

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

1.3 Structure of the EDT statements manual

This manual first provides an introduction to EDT, followed by a description of files and library elements as well as the use and creation of EDT procedures. It also provides an overview of all the EDT statements together with a detailed description and a large number of examples.

The extensions to the compatibility mode and the way it interacts with Unicode mode are presented in a separate chapter.

This manual covers the following individual topics:

– **Modified and new functionality in EDT V17.0A**

  major changes and important new features in EDT V17.0A
  Introduction to the new functionality.

– **Underlying EDT concepts**

  The fundamental concepts and mechanisms on which EDT is based.
  This includes the handling of work files, operations involving lines, character sets and EDT variables as well as the use of EDT procedures.

– **Running EDT**

  The EDT start command. Starting, interrupting and terminating an EDT session.
  Monitoring the EDT session, input/output, job switches and access protection.

– **EDT work modes**

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

– **File processing**

  Processing of all the file types supported by EDT: ISAM, SAM, POSIX files and libraries.

– **EDT statements**

  A thematically organized overview of the statements. Presentation of the metasyntax, statement syntax and operand syntax.
  EDT statements in alphabetical order accompanied by numerous examples.

  In many statements, the subdivision and designation of the formats is now clearer than the presentation in the EDT V16.6B manual. However, these changes are simply modifications to the presentation which do not reflect any technical differences.

– **Compatibility mode**

  Description of the extensions to the compatibility mode and the way it interacts with Unicode mode. New statements in compatibility mode.

– **EDT messages**

  List of all the EDT messages together with their meanings and the actions to be taken in response to them.

– **Logistics**

  Requirements and procedures for the installation and start-up of EDT V17.0A.

2 Modified and new functionality in EDT V17.0A

EDT version V17.0A provides a range of important new functions.
For example, it supports the processing of files coded in Unicode and the restriction of record lengths to 256 characters has been eliminated.
A brief introduction to the modified and new functionality is provided below.

2.1 Introduction to the EDT operating modes

The most important new feature of EDT V17.0A compared to the preceding versions is the capability to process files coded in Unicode. This support was introduced in order to achieve the following two aims:

- Users who want to process files coded in Unicode should be able work in an easy-to-use environment. This includes, for example, the ability to edit differently coded files in various EDT work files simultaneously as well as the elimination of the line length restriction (previously 256 characters).
- Users who want to continue to edit files coded in 7-bit or 8-bit character sets as before should be able to make use of functions and interfaces compatible with EDT V16.6B. This relates, in particular, to the execution of EDT procedures and operations at the subroutine interfaces.

The increase in the permitted record length together with the fact that a Unicode representation is used internally in the work files means that the compatibility of all the various interfaces at which users previously had direct access to the internal EDT data cannot be maintained. This applies to the old L mode subroutine interface, the former @RUN interface and the Locate mode of the IEDTGLE interface. It is therefore no longer possible to use these interfaces if you wish to make use of the new functionality.

Nevertheless, programs which simply use the EDT subroutine interfaces to permit their users to edit files without having to exit the current program should also be able to operate with Unicode files with the smallest possible number of changes.

This is possible because it is now possible to run EDT V17.0A in two modes:

- In Unicode mode which has been extended for the processing of Unicode files but in which there are certain incompatibilities, in particular at the level of the subroutine interface.
- In compatibility mode which offers the full functionality of EDT V16.6B but which does not support the processing of Unicode files, increased record lengths or locally configured character sets.

For more information on the **operating modes** and the way they **interact** see **chapter 11**.

2.2 Unicode mode

2.2.1 Additional functions - overview

In Unicode mode, the following additional functions are available:

- Processing of files with different character sets in the EDT work files, and in particular files with Unicode coding (`UTF16`, `UTFE` or `UTF8`) or ISO coding (the special treatment of POSIX files coded in ISO is eliminated in the new approach). Interpretation of a substitute representation of Unicode characters.
- Processing of files with record lengths of up to 32768 bytes (upper limit of DMS).
- Consistent handling up empty records.
- All 23 work files are also available in F mode.
- Consistent extension to many statements.

2.2.2 Additional functions - explanations

Local character sets

In EDT V17.0A Unicode mode, it is possible to define a different character set for each work file. This includes the 7-bit and 8-bit character sets which were supported in the past as well as the Unicode character sets UTF8, UTF16 and UTFE, the ISO character sets supported by XHCS and the user-defined character sets declared in XHCS.

In statements both in literals and in data, it is possible to specify Unicode characters by means of a substitute representation by defining the hexadecimal value of the corresponding UTF16 code.

The character sets for the individual work files are set either implicitly by reading a correspondingly coded file or explicitly via the @CODENAME statement.

A detailed description of working with local character sets, and in particular of data transfer between work files or between EDT variables and work files can be found in the section "Character sets" on page 47.

Long records

The restriction of the record length to 256 characters is eliminated in EDT V17.0A Unicode mode. When reading from and writing to DMS files, EDT is able to process records of a maximum length of 32768 bytes (depending on file format). This limit is imposed by the DMS and refers to the *byte count*. In the case of library elements, the limit imposed by LMS is 32763 bytes. Because when Unicode character sets are used, characters may be coded by multiple bytes, the permitted number of *characters* per record may be lower. Internally, EDT works with long buffers with the result that this restriction only has an impact when the records are converted into the work file character set for write operations. EDT provides a statement (@CHECK, Format 2) which can be used to check whether records need to be truncated on write operations.

In the case of POSIX files, the line length is restricted only by the maximum EDT line length of 32768 characters (not bytes).

The elimination of the record length restriction applies equally to records in work files, string variables and statement lengths and has an effect on the statement syntax (e.g. in the case of column specifications), the layout of the status display, the subdivision of the screen on @EDIT LONG and on the subroutine interface.

Consistent handling of empty lines

The files that are to be processed by EDT may contain records of length 0.
In the case of POSIX or SAM files, the records genuinely have length 0. In the case of ISAM files with standard attributes, the records may have the length 8 or 16 (in the case of files coded in UTF16).

To permit the depiction of records of length 0 in the data window, EDT in Unicode mode indicates the end of the record using a terminal-specific logical line end character LZE. The terminal fills the remainder of the screen to the right of LZE with protected NULL characters (X'00'). If the end of the record is located outside of the data window then LZE is not displayed. The screen line then ends with the last character of the record that is still visible or consists only of protected NUL characters.
A record of zero length is therefore depicted in the data window by a screen line which consists only of the character LZE in column 1 and protected NULL characters (empty line). If LZE is entered in column 1 of a line then a record of length 0 is created for this line in the work file.

Empty lines should be distinguished from new lines which EDT provides in F mode after the last record in the file or during the processing of the statement codes 1..9 or I. These lines do not (yet) correspond to any record in the work file and consist only of NULL characters (X'00') without LZE (and which can be overwritten).

The LZE character can usually be omitted during input.
It only has to be entered when the record is intended to end with NULL characters. When input is performed in F mode, EDT ignores all NULL characters at the end of the entered line, i.e. all the NULL characters up to the first character which is not NULL (this can be an LZE or another character) are truncated from the right. The LZE itself is ignored on input. Since new lines only consist of NULL characters they are ignored overall on input and are not inserted in the work file.
In contrast, entering an LZE in column n of a new line would cause a record with n−1 blanks to be inserted by default in the work file after data transfer (or alternatively a record with n−1 NULL characters depending on the definition made using @SYMBOLS FILLER). In particular, a record of length 0 would be inserted for n=1.

The terminal does not permit any entries to the right of the LZE character in a line. When adding entries to a line, it is therefore necessary to activate the terminal insertion mode or to overwrite the LZE character. This incompatibility with EDT V16.6B is tolerated since it permits the consistent handling of blank lines in Unicode mode.

For a more precise description of program behavior on the entry of lines which contain NULL characters or fill characters, see section “F mode” on page 101.

Availability of all work files

In F mode, it is now possible to switch to one of the work files 10 to 22 by entering the corresponding number in the statement line. In the past, this was only possible in L mode with the help of the @PROC statement (see also section "F mode" on page 101).

Uniformity of the statement interfaces

In EDT V17.0A, it is now possible to access files of all supported file types using a uniform set of statements (@OPEN, @CLOSE, @COPY, @WRITE) (see also "File processing" on page 131 and the descriptions of the individual statements).

To make this possible, the statements have been extended by the corresponding operands. Although it is still possible to use the old statements, users are recommended to employ only the new statements.

For reasons of completeness and consistency, new operands have also been introduced for other statements. For the related details, see the statement descriptions.

2.2.3 Functions that are no longer supported

In Unicode mode, the following EDT V16.6B functions are no longer available:

- Output to SDF list variables on @LOG
- V15-compatible L mode syntax control
- Support for the old L mode subroutine interface
- Support for the `IEDTGLE` interface's Locate mode.
- Support for the previous @RUN interface. A new @RUN interface is now available.
- Support for terminals with Arabic or Farsi character sets.
- Support for 3270 terminals (IBM) and printer terminals.
- The @CODE statement whose use in combination with XHCS was already indicated as pointless in the EDT V16 manual.
- The @UPDATE and @ZERO-RECORDS statements.

2.3 Compatibility mode

The compatibility mode provides the full functionality of EDT V16.6B including the old L mode subroutine interface. The extended functions provided in Unicode mode are not available in compatibility mode.

The EDT V17.0A compatibility mode has simply been extended by a small number of required functions.

For a detailed description of **compatibility mode** see **chapter 11**.

2.3.1 @CODENAME statement

The @CODENAME statement has also been extended in compatibility mode. This means that it is possible to perform targeted modifications to the character set for migration purposes even in non-empty work files.

2.3.2 @IF statement

The EDT V17.0A compatibility mode has been extended by format 5 of the @IF statement. This makes it possible to query the current operating mode and react if necessary.

2.3.3 @MODE statement

The new @MODE statement is also present in compatibility mode and makes it possible to switch to Unicode mode.

2.3.4 Messages

The message `EDT4983` may occur in EDT V17.0A compatibility mode.

3 Underlying EDT concepts

This section describes the underlying concepts and high-level mechanisms which are used in EDT.

3.1 Work files

Users have 23 work files in virtual memory available to them for file processing purposes. These are work files 0 to 22. The work files are able to accommodate records of a length up to 32768 characters. This means that it is possible to process DMS files and library elements with the maximum permitted record length. The maximum number of records permitted in a work file is 99999999.
In work files, it is possible to enter new data, read in existing files for processing, generate and edit data using EDT statements or copy data from other work files.

In F mode, work file 9 is used by a number of EDT statements in order to store results (@COMPARE, @FSTAT, @SHIH, @SHOW, @STAJV, @STATUS).
This may cause existing content to be deleted without warning.

However, if a file is open in work file 9 then a message `EDT5189` is output and the related statement is not executed. Work file 9 should therefore only be used as a temporary help file.

3.1.1 Properties of work files

Each work file has certain properties which can be modified using EDT statements and which have an effect on the operation of EDT statements or the way work files are displayed. The table below collates the various work file properties.

Properties	Initial value	Value can be changed by
General		
Current character set for the work file	*NONE	@CODENAME implicitly via data input (file read operations, screen input, statements)
Work file occupied (only for work files 1 to 22)	not used	@PROC, @DELETE, @DROP, used in F mode
Work file empty	Yes	Read file, screen input, miscellaneous statements
Work file modified	No	Modify work file, @DELETE
Save file present	No	@AUTOSAVE and modify work file, @DELETE
Line numbers		
Current line number (symbolic line number *)	1.	@SET (Format 6), @+, @-, implicitly via data input (read file, screen input, statements)
Current increment value	1.	@SET (Format 6), @PAR INCREMENT
Current renumbering value	Off	@PAR RENUMBER
Lowest assigned line number (display by means of @LIMITS)	0.0000	Implicitly due to file read operation, screen operation, miscellaneous statements
Symbolic line number %	= *	Implicitly due to file read operation, screen operation, miscellaneous statements
Highest assigned line number (display by means of @LIMITS)	0.0000	Implicitly due to file read operation, screen operation, miscellaneous statements
Symbolic line number $	= *	Implicitly due to file read operation, screen operation, miscellaneous statements
Symbolic line number ?	0.0000	@ON
Memory area @SET (Format 6)	Empty	@SET (Format 6)

Properties	Initial value	Value can be changed by
File processing		
Link to file name (local @FILE entry)	No link	@FILE, @READ, @GET, @DELETE
Link to open file	No link	@OPEN, @CLOSE
Default library name	*NONE	@PAR LIBRARY
Default element type	S	@PAR ELEMENT-TYPE
Default character set for a POSIX file	EDF041	@PAR CODE
Input		
Differentiation between uppercase and lowercase	Off	@PAR LOWER
Maximum record length for entry in F mode	32768	@PAR LIMIT
Escape character for Unicode substitute representation	*NONE	@PAR ESCAPE-CHARACTER
Unicode substitute character including for data	Off	@PAR DATA-REPLACEMENT
Representation of work file		
Full display of records in F mode	Off	@PAR EDIT-LONG
Hexadecimal mode	Off	@PAR HEX
Data window and statement code column are both overwritable simultaneously	Off	@PAR EDIT-FULL
Ruler in data window	Off	@PAR SCALE
Information line in data window	Off	@PAR INFORMATION
Data window-specific representation		
First line displayed in data window 1	0.0000	@SETF, +, −, ++, −−, statement codes +, −, B, I, S, 1..9 in data window 1
First column displayed in data window 1	1	@SETF, >, <, << in data window 1
Line number display in data window 1	Off	@PAR INDEX in data window 1

Properties	Initial value	Value can be changed by
First line displayed in data window 2	0.0000	@SETF, +, −, ++, −−, statement codes +, −, B, I, S, 1..9 in data window 2
First column displayed in data window 2	1	@SETF, >, <, << in data window 2
Line number display in data window 2	On	@PAR INDEX in data window 2
Other		
Program name for SDF syntax check	*NONE	@PAR SDF-PROGRAM
Type of program name for SDF syntax check	INTERNAL	@PAR SDF-NAME-TYPE
Character for data record separation	*NONE	@PAR SEPARATOR
Character for structure sheets	@	@PAR STRUCTURE
Write protection at record level	Off	@PAR PROTECTION
Indicator for hits on last @ON for @IF (Format 3)	Off	@ON
Column for hits on last @ON for @IF (Format 3)	0	@ON

The work file properties defined with the @PAR statement can be reset to their initial values using @PAR $0..$22 without the need for any further operands.

3.1.2 Current work file

At any time there is precisely one work file which is referred to as the current work file. Data is entered and EDT statements are executed in the current work file if no other work file is explicitly specified in the statement).

In F mode, a section of the current work file is usually displayed on the screen. The number of the current work file is displayed in the status bar. It is possible to move the displayed section of the work file (see the statements @SETF and +, -, ++, --, >, <, <<) or change the current work file (see statements @SETF, $0..$22 and 0..22).

It is also possible to divide the work window and display sections from two work files simultaneously (see the section on the F mode). In this case, the current work file switches between the two displayed work files. When a statement or statement code is processed (see section "Processing sequence" on page 115) the current work file is always considered to be the work file containing the statement line or statement code column from which the current statement or statement code comes. When input in the data window is processed, the current work file is considered to be the work file in whose data window the input is made.

In L mode, the number of the current work file can be displayed by means of the @PROC statement. The current work files are switched using the statements @SETF, @PROC and @END. However, an active work file (which contains an active @DO procedure) can never be made into the current work file.

When EDT is started, work file `0` is the current work file.

3.1.3 Empty work file

An empty work file is a work file which contains no records. This is generated when

– EDT is started or

– when the work file is completely deleted with @DELETE (Format 2), @DROP or other statements which implicitly execute a @DELETE (Format 2) or

– in F mode, when all the lines are deleted with the statement code `D` or `M` or when all the lines are deleted with @DELETE (Format 1), @MOVE or @ON (Format 8 or 10).

In dialog operation, an empty work file can be recognized by the fact that it contains no records. In EDT procedures, it is possible to use the @IF statement (Format 3) to check whether a work file is empty.

It is possible to write an empty work file. In such cases, the character set configured for the work file may also be configured for the file depending on the operands set for the relevant statement.

The properties of a work file when EDT is started are indicated in the table in the preceding section. All the work file properties defined with the @PAR statement, with the exception of those explicitly listed below, are retained following a delete operation.

The table below collates the additional properties of a work file following deletion.

Properties	Values after deletion with @DELETE (Format 2) or @DROP	Values after the deletion of all the present lines
General		
Current character set for the work file	*NONE	Not changed
Work file occupied (only for work files 1 to 22)	None (apart from current)	Yes
Work file empty	Yes	Yes
Work file modified	No	Yes
Save file present	Save file deleted	Save file deleted
Line numbers		
Current line number (symbolic line number *)	1.	0 + Current increment value
Current increment value	1.	Not changed
Lowest assigned line number (display by means of @LIMITS)	0.0000	0.0000
Symbolic line number %	1.	==*
Highest assigned line number (display by means of @LIMITS)	0.0000	0.0000
Symbolic line number $	1.	==*
Symbolic line number ?	0.0000	Not changed
Memory area (Format 6)	Empty	Not changed
File processing		
Link to file name	No link	Not changed
Link to open file	No link (implicit @CLOSE executed)	Not changed (can be revoked with @CLOSE)
Input		
Unicode substitute character including for data	Off	Not changed

Properties	Values after deletion with @DELETE (Format 2) or @DROP	Values after the deletion of all the present lines
Data window-specific representation		
First line displayed in data window 1	0.0000	0.0000
First column displayed in data window 1	1	Not changed
First line displayed in data window 2	0.0000	0.0000
First column displayed in data window 2	1	Not changed
Other		
Indicator for hits on last @ON for @IF (Format 3)	Off	Not changed
Column for hits on last @ON for @IF (Format 3)	0	Not changed

3.2 Line numbers

Every line in a work file has an 8-digit line number (range of values 0000.0001 to 9999.9999). However, there does not have to be a line for every line number.
The line numbers permit the unique identification of the lines in a work file. In addition, the numerical values of the line numbers define a sequence for all the lines in a work file (if a line has a lower number than another then it must precede the other line).
This means that it is possible to specify line ranges by defining the lowest and highest line number in the range.
In many EDT statements which process lines in whatsoever form, line numbers are used in order to define the lines or line ranges that are to be processed by the statement.

3.2.1 Current line number and current increment

A *current line number* and *current increment* are assigned to each work file. In L mode, data is entered in the line with the current line number.
The new, current line number is then determined on the basis of the previous current line number plus the current increment value. If `WRTRD` is used to perform a read operation in L mode then the current line number is used as a prompt for data entry. A line with the current line number may exist but this is not obligatory.

In some EDT statements, the location in the work file at which the lines are to be inserted can be defined by means of a temporary current line number and a temporary current increment which are specified using the operands of the corresponding EDT statement and which are valid only in this single statement or, otherwise, assume the role of the current line number and current increment in this EDT statement. The current line number can also be changed by running this type of EDT statement.

When EDT is started or after a work file has been completely deleted (either explicitly or implicitly), the current line number is `1.0000` and the current increment is `1.0000`. The current line number and the current increment can be redefined using the @SET statement (Format 6) or the current increment can be set separately by means of @PAR INCREMENT.

The statements @+ and @- are used to redefine the current line number by adding or subtracting the current increment to or from the previous current line number. If the @EDIT statement has first been issued with the `SEQUENTIAL` operand then the current line number is only formed in this way if there are no lines between the previous current line number and the new current line number. If this is not the case, the first intervening line number becomes the current line number. In all cases, the current increment remains unchanged.

The @RENUMBER statement modifies both the current line number and the current increment. The new current line number is the line number of the last line in the work file after renumbering plus new current increment defined by means of the @RENUMBER statement. If only a line number is specified in the @RENUMBER statement then the new current increment is implicitly defined in the same way on the basis of this line number as, for example, in the @SET statement, format 6 (see section “Implicit increment assignment” on page 35).

The current line number and the current increment are also modified by all statements which completely delete the work file either explicitly or implicitly. After the explicit or implicit deletion to the complete work file, the current increment is `1.0000` and the current line number is `1.0000`. However, if the entire work file is implicitly deleted, the current line number is then usually modified again by the application responsible for deletion.

If the @SET statement (format 6) is used to modify the current line number and the current increment then the pair of values consisting of the previous current line number and the previous current increment is saved in a memory area which may contain a maximum of three such pairs of values (see the description of the @SET statement, format 6).

If the @SET statement (format 6) is specified without parameters then the last pair of values saved in this memory area becomes the current line number and current increment again. All the statements which explicitly or implicitly delete the work file also completely delete this memory area.

3.2.2 Symbolic line numbers

Alongside the line number variables (#L00..#L20) which can be assigned any line numbers, there are also four special characters (*, %, $, ?) which represent symbolic line numbers.
The current line number can be addressed via the symbolic line number *, the lowest line number in a work file by the symbolic line number % and the highest line number in a work file by the symbolic line number $.

When EDT is started, these three symbolic line numbers have the value 1.0000. The symbolic line number ? contains the first hit line returned by a preceding @ON statement. When EDT is started, it has the value 0.0000 and is only modified by @ON statements which return a hit. The four symbolic line numbers *, %, $ and ? are work file-specific.
In addition, the @DO statement (format 1) makes it possible to define a special character as a loop character in a @DO procedure. This loop symbol is assigned a sequence of values which are defined by means of a start value, an end value and an increment (which must also be specified in the @DO statement).
In the procedure, this special character can then be used like a symbolic line number representing the current value of the loop symbol in EDT statements. For the special characters that can be used as the loop symbol, see the description of the @DO statement (format 1).

3.2.3 Implicit increment assignment

If, in the case of statements for which both a line number and an increment can be specified (e.g. @SET, format 6), only a line number is specified then the increment is implicitly defined by the specified line number (the line number and increment may be either the current line number and current increment or may take the form of specifications which apply only to the statement in question).
If the expression which defines the line number contains only line number variables (#L0..#L20) or symbolic line numbers (%,*,$,?) or integer variables (#I0..#I20) or relative line number specifications (nL) or a combination of these then the new current increment is 0.0001.
If a numerical line number specification forms part of a statement then the number of decimal places in this numerical line number specification (maximum 4) determines the current increment. If no decimal place is present then the current increment is 1, if one decimal place is present then the current increment is 0.1 etc.
If four decimal places are present, the current increment is 0.0001.

3.2.4 Line number assignment

Both the inclusion of new lines in a work file by means of EDT statements (e.g. by reading a file or copying lines etc.) and the deletion of lines if the previous last line in the work file is one of the deleted lines usually result in a modification to the current line number (there are a few exceptions where the current line number remains unchanged). The new current line number is usually determined from the line number of the last line in the work file plus the current increment. However, in some cases, the current line number is the line number of the last line inserted by the EDT statement plus the current increment.

The numbering of lines which are newly inserted in a work file is performed using one of the five procedures listed below.

3.2.4.1 Using the source line numbers

The line numbers in the source are used. In the case of a copy operation, these are the lines to be copied from another work file.
When data is read from a file then, in the case of ISAM files, the line numbers are formed from the ISAM key (unless specified to the contrary). In the case of SAM files with the `KEY` operand specified, the first eight characters of each record are interpreted as the line number.
Any lines with the same line numbers present in the target file are overwritten. The new current line number is formed from the line number of the last line plus the current increment if the line number of the last line has changed. Otherwise it remains unchanged.

The following table indicates the EDT statements for which this procedure is used (in the case of some EDT statements, other procedures may be used in the event of a different format or different operands).

Statement	Comments on operands	General comments
@COPY Format 1	Read in ISAM files with operand specification KEY=LINENUMBER	
@COPY Format 2	No target specification	
@GET	With operand specification NORESEQ	
@MOVE	No target specification	
@ON Format 4 + 5	With operand specification KEEP	If the OLD operand is not specified then the target work file is deleted before insertion.
@OPEN Format 1	Open ISAM files with operand specification KEY=LINENUMBER	The work file must be empty before the statement is executed.
@OPEN Format 2	Open a copy of SAM files with specification of KEY operand and open a copy of ISAM files	The work file must be empty before the statement is executed. If a file is already opened with the @OPEN statement (format 2) and a new @OPEN statement (format 2) is issued then the second @OPEN statement is implicitly preceded by a @CLOSE statement, i.e. the work file is implicitly deleted.
@READ	With specification of the KEY operand	

3.2.4.2 Insertion at the current line number

Lines are inserted starting at the current line number. The line numbers of the other lines that are to be inserted are determined by adding the current increment to the line number of each last inserted line. Any lines with the same line numbers present in the target file are overwritten. The new, current line number is then the line number of the last newly inserted line plus the current increment.

The following table indicates the EDT statements for which this procedure is used (in the case of some EDT statements, other procedures may be used in the event of a different format or different operands).

Statement	Comments on operands	General comments
L mode data entry		This not only relates to input at the L mode prompt but also to data inputs in EDT procedures and via the `text` operand of some L mode statements.
@GET	No operand specification `NORESEQ`	
@GETLIST		
@ON Format 4 + 5	No `KEEP` operation specified but `OLD` operand specified	
@OPEN Format 2	Open a copy of SAM files without specifying the `KEY` operand	The work file must be empty before the statement is executed. This is the case after EDT is started or after an explicit deletion. In both cases, the current line number and the current increment are both `1.0000`. However, it is possible to set a new current line number and a new current increment before executing the statement.
@READ	Without specification of the `KEY` operand	

3.2.4.3 Insertion after implicit deletion

The work file in which the lines are to be inserted is implicitly completely deleted prior to insertion, i.e. the current increment and the current line number are both `1.0000` immediately before insertion. Lines are then inserted starting at the current line number(`= 1.0000`).

The line numbers of the other lines that are to be inserted are determined by adding the current increment (`= 1.0000`) to the line number of each last inserted line. On completion of the insertion operation, the new current line number is determined from the line number of the last newly inserted line plus the current increment.

The following table indicates the EDT statements for which this procedure is used (in the case of some EDT statements, other procedures may be used in the event of a different format or different operands).

Statement	Comments on operands	General comments
@ON Format 4 + 5	Without specification of the `KEEP` and `OLD` operands	
@OPEN Format 2	Open a copy of SAM files without specifying the `KEY` operand in a work file in which a file has already been opened with @OPEN (format 2).	Before the second @OPEN statement, a @CLOSE statement is issued, i.e. the work file is implicitly deleted.
@SHIH		The output is sent to work file 9.
@STATUS	Without target specification on output in F mode	The output is sent to work file 9.

3.2.4.4 Insertion at predefined line number

Lines are inserted starting at the line number specified as an operand in the statement. The line numbers of the other lines that are to be inserted are determined by adding any increment specified as an operand in the statement to the line number of the last line to be inserted.

If no increment is specified in the statement then it is defined implicitly on the basis of the specified line number (see section "Implicit increment assignment" on page 35) using the same procedure as for implicitly defining the current increment on the basis of the current line number. Any lines with the same line numbers present in the target file are overwritten. The new current line number is formed from the line number of the last line plus the current increment if the line number of the last line has changed. Otherwise it remains unchanged.

The following table indicates the EDT statements for which this procedure is used (in the case of some EDT statements, other procedures may be used in the event of a different format or different operands).

Statement	Comments on operands	General comments
@COMPARE Format 1	If the LIST operand is not specified, output is sent to the screen; if LIST is specified without a line number then it is sent to SYSLST	
@COPY Format 2	With target specification	
@FSTAT	With target specification	
@GETJV	With output to a line	Only one line is generated.
@GETVAR	With output to a line	Only one line is generated.
@MOVE	With target specification	
@SHOW Format 1 +2	With target specification	
@STAJV	With target specification	
@STATUS	With target specification	
@SYSTEM	With target specification	

3.2.4.5 Insertion between two lines

In this procedure, the new lines for insertion are inserted between two existing lines without overwriting any existing lines. The procedure first attempts to insert the new lines on the basis of the current increment. If it is not possible to insert the new lines in this way, the increment used for line insertion is repeatedly divided by 10 until it is possible to insert the lines or the smallest possible increment of `0.0001` has been reached.

If it is still not possible to insert the new lines in full at the smallest possible increment of `0.0001` then the insertion procedure is interrupted with an error message if the @PAR RENUMBER=OFF statement has previously been used to deactivate automatic line renumbering.

If, however, automatic line renumbering is active (@PAR RENUMBER=ON), then the procedure attempts to renumber the larger of the two line numbers between which insertion is to be performed and any other line numbers following this in such a way that the insertion operation can be completed successfully.

Only if it is not possible to renumber sufficient lines is the insertion operation interrupted with an error message. When EDT starts, the setting @PAR RENUMBER=ON always applies for all work files. However, the setting can be modified individually for each work file.
The insertion operation is described in rather more detail below.

In this insertion procedure, it is always necessary to have two line numbers between which the new lines are to be inserted. One of the two line numbers is predefined (in some cases implicitly) by the corresponding statement. In the case of statement codes in F mode this may be, for example, the line number of the line in which the statement code was entered, thus in the case of the @XCOPY statement, for example, the line number of what was previously the last line in the work file.

In most cases, insertion is performed after this explicitly or implicitly specified line number. The line number of the next line is then used as the second line number. If insertion is performed after the last line in the work file then the value `10000.0000` is used as the fictitious line number for the second line.

Only in the case of the statement codes `B,I,1..9` and the @COPY statement (format 1) with the `BEFORE` parameter specified is insertion performed before the predefined line. In this case, the line number of the preceding line is used as the second line number.
If insertion is performed before the first line then the value `0.0000` is assumed for the second line. If the work file is empty then `0.0000` is assumed for the first line number and `10000.0000` for the second line number.
In this case, the fictitious line number `0.0000` is used to determine the line number of the first line that is to be inserted (= `0.0000` + increment) and cannot be occupied itself.

Insertion without automatic renumbering (@PAR RENUMBER=OFF)

The current increment is used as the increment for the first insertion attempt. The line number of the first line for insertion is obtained by adding the increment to the smaller of the two line numbers between which insertion is to be performed. The line numbers of the following lines for insertion are obtained by continuing to add the increment to each new value calculated. If the lines numbered in this way fit between the two starting lines then this numbering is used.
However, if they do not fit then the subsequent procedure depends on whether or not the increment is equal to its smallest possible value of `0.0001`.
If it is equal to `0.0001` then the insertion of lines is rejected with the message `EDT5365`.
If the increment is greater than `0.0001` then it is used to calculate a new increment by dividing the preceding increment by 10 (the current increment is not modified!). If this new increment is smaller than `0.0001`, then `0.0001` is used as the new increment.
This new increment is then used for a new insertion attempt. If the lines for insertion can be inserted using this new increment then the operation is complete.
However, if the increment is already `0.0001` then the insertion operation is rejected with the message `EDT5365`. If the increment is still greater than `0.0001`, then it is again divided by 10 and a new attempt is made with this new increment.
The current line number only changes if at least one new line is created with a line number greater than the previous highest line number and is then equal to the line number of the last line in the work file plus the current increment (this is only possible when a line is inserted after the previous last line).

Insertion with automatic renumbering (@PAR RENUMBER=ON)

In this case, a two-stage procedure is used in order to maintain maximum compatibility with EDT V16.6B.
Exactly the same procedure is used as for @PAR RENUMBER=OFF.

If the increment has reached the value `0.01` and the lines can still not be inserted then an attempt is made to renumber all the existing lines which would be overwritten by the newly inserted lines in such a way that they can be appended at the last line inserted so far with an increment of `0.01`. This is the procedure used in EDT V16.6B or in compatibility mode.
If it proves impossible to renumber sufficient lines in this way, EDT V16.6B or the compatibility mode interrupts the insertion operation with an error message.
In contrast, Unicode mode attempts to further reduce the increment (i.e. division by 10) down to the smallest increment value of `0.0001` in order to insert the lines.
Once the smallest possible increment of `0.0001` has been reached, the lines are always inserted with this increment of `0.0001`. All the existing lines which would be overwritten by the newly inserted lines are now renumbered in such a way that they are appended to the previous last inserted line using an increment of `0.0001`.

If this renumbering again overwrites existing lines then the operation is repeated, i.e. these lines are also renumbered. If the renumbering operation does not overwrite any more existing lines then the insertion/renumbering procedure has been completed.
Here again, the current line number only changes if at least one new line is created with a line number greater than the previous highest line number and is then equal to the line number of the last line in the work file plus the current increment.

The statement code `0` represents an exception here.
When this statement code is issued, only existing lines are overwritten starting with the line indicated by `0`. If all the lines following the line indicated by `0` have already been overwritten and there are still lines present for insertion then these lines are inserted after the last overwritten line in accordance with the procedure described above.

The following table indicates the EDT statements for which this procedure is used (in the case of some EDT statements, other insertion procedures may be used in the event of a different format or different operands). Statements which do not start with an EDT statement symbol are statement codes.

Statement	Comments on operands	General comments
A		
B		
I		
1..9		
O		See above for differences from usual procedure
T		
@COMPARE Format 2		The work file is deleted before use and insertion is therefore performed after line number `0.0000`.
@COPY Format 1	Read in SAM files, POSIX files and library elements as well as ISAM files with the `KEY` operand being specified and not equal to `LINENUMBER`	If neither the `AFTER` nor the `BEFORE` operand is specified then insertion is performed after the previous last line; otherwise before or after the specified line.
@FSTAT	Without target specification on output in F mode	The work file `9` is deleted before use and insertion is therefore performed after line number `0.0000`.

Statement	Comments on operands	General comments
@OPEN Format 1	OPEN SAM files, POSIX files and library elements as well as ISAM files with the KEY operand being specified and not equal to LINENUMBER	The work file must be empty before the statement is executed and insertion is therefore performed after the line number 0.0000.
@SDFTEST		
@SEPARATE		
@SHOW Format 1 +2	Without target specification on output in F mode	The work file 9 is deleted before use and insertion is therefore performed after line number 0.0000.
@STAJV	Without target specification on output in F mode	The work file 9 is deleted before use and insertion is therefore performed after line number 0.0000.
@XCOPY		If the work file is not empty, insertion is performed after the previous last line. If the work file is empty, it is performed after line number 0.0000.
@XOPEN		The work file must be empty before the statement is executed and insertion is therefore performed after the line number 0.0000.

3.3 Record marks

Every line in an EDT work file can be flagged with one or more record marks. The record marks are noted in the EDT data area and are not visible to users. They are not taken over when files are written to.
If EDT is not called as a subroutine, record marks 1 to 9 are available. If EDT is called as a subroutine, the record marks 13, 14 and 15 can also be used for special functions.
The record marks 0, 10, 11 and 12 are reserved for internal special functions and are not available to users irrespective of whether or not EDT is called as a subroutine.

Record marks 1 to 9 can be set and deleted both with EDT statements and statement codes and with the `IEDTPUT` and `IEDTPTM` functions provided by the EDT subroutine interface. In contrast, record marks 13, 14 and 15 can only be set and deleted with the `IEDTPUT` and `IEDTPTM` functions.

No marks can be entered for lines in ISAM files opened for real processing with the @OPEN statement (format 2).

Record marks can be set using

– the statement code `1..9` F3 in the statement code line
– the @ON statement (format 3) or
– the `IEDTPUT` and `IEDTPTM` functions when EDT is called as a subroutine

Record marks can be deleted using

– the statement code `D` F3 in the statement code line
– the @DELETE statement (format 4) or
– the `IEDTPUT` and `IEDTPTM` functions when EDT is called as a subroutine

In F mode, record marks 1 to 9 can be used as the target for positioning in the work file, either by means of one of the statements `+`, `++`, `-` or `-` followed by F3 or with the @SETF statement.

In the case of the @ON (format 4) and @SETLIST statements with the `MARK` parameter, record marks 1 to 9 are used to select the lines that are to be processed.
The @ON statement (format 4) can be used to copy marked lines to another work file while @SETLIST can be used to take over the marked lines into an SDF-P list variable.

The record marks are deleted if a new record is created and replaces another record. However, they are not deleted if a record is simply modified. New records are created using the statements: @CREATE, @COPY (format 2), @MOVE, @READ, @GET, @GETJV, @GETVAR, @GETLIST @ON (format 4, 5) and the statement codes `A`, `B` and `O`. Modifications consist of the input of new content at the terminal (typed input), changes caused by the statement codes `L` and `U` and the statements @PREFIX, @SUFFIX, @CONVERT, @COLUMN, @SEQUENCE (format 1, 2) and @ON (format 6, 7, 8, 9). In the case of the

@SEPARATE statement, the original line retains the mark but the newly generated lines are not marked. In the event of a join with J, the top line retains any mark it may have had but the mark is not taken over by the lower line.

When renumbering is performed with @RENUMBER and @SORT or implicit renumbering results from the insertion of other records, the marks in a record are retained.

If EDT is used as a subroutine, record marks can be set and deleted using the IEDTPUT and IEDTPTM functions.

The IEDTGTM function can be used to read lines with specific record marks. A line's record mark is also returned when reading is performed with IEDTGET.

Record mark 13 has the special function of an ignorer indicator. Lines marked in this way are

- automatically deleted on return of control to the main program after a @DIALOG call via the subroutine interface
- not included when writing to a file or library element
- not copied when lines are copied
- only taken into consideration by the IEDTGET and IEDTPUT functions if the flag EAMIGN13 is set in the EAMFLAG field of control block EDTAMCB. The functions IEDTGTM and IEDTPTM always take account of record mark 13 irrespective of whether the flag EAMIGN13 is set in the EAMFLAG field.

In the @SDFTEST statement or the statement code T as well as in the statement codes J, C, M, R, A, B and O, lines with record mark 13 are always ignored.

Record mark 14 has the special function of an update indicator. Lines marked in this way are depicted as being available to be overwritten in F mode. They continue to be available for overwriting if only DUE is sent or if this type of line is modified by a statement (e.g. the @ON statements, formats 6 to 9).
Only if at least one character is entered directly in the line and sent with DUE is the record mark 14 deleted, with the result that the line is no longer displayed as being available for overwriting.

Record mark 15 has the special function of a write protection indicator. Lines with record mark 15 cannot be set to overwritable with the statement codes X, H or with F2 in the F mode screen dialog.

However, such lines can be modified using statements (e.g. the @ON statement, formats 6 to 9) or one or more following lines can be appended to a line with record mark 15 using statement code J
In both cases, record mark 15 continues to be present.

PROTECTION=ON must be set using the @PAR statement before it is possible to evaluate record marks 14 and 15.

If a record with one of the marks 13, 14 or 15 is modified at the terminal by means of an entry in the data window (typing) then these special marks are deleted.

3.4 Character sets

EDT makes it possible to process texts present in different character sets. In Unicode mode, data can and must be converted between character sets. This greatly extends the functionality compared to compatibility mode (see chapter “Compatibility mode” on page 611).

It is possible to specify a different character set for each work file. It is therefore possible to process data in different character sets in different work files. These character sets can be modified at any time using the @CODENAME statement. In addition, EDT possesses its own character set – the communication character set – which it uses to communicate with the terminal. This can be different from the character set used in any work file in which data is stored.

3.4.1 Character sets in BS2000

In BS2000, character sets are provided by the software product XHCS. By default, these include:

– 7-bit character sets such as, for example. `ISO646` (international 7-bit character set, ASCII), `EDF03IRV` (international reference version, EBCDIC), `EDF03DRV` (German reference version, EBCDIC).
– 8-bit character sets such as, for example. `ISO88591` (Latin Alphabet No.1, ASCII), `EDF041`
(Latin Alphabet No.1, EBCDIC), `EDF04DRV` (extension of `EDF03DRV`) etc.
– The 3 Unicode character sets `UTF16`, `UTF8` and `UTFE`.

XHCS also makes it possible to provide user-defined character sets. These character sets must be assigned all the attributes that the character sets defined by default also possess, i.e. all the property tables must be present. If this is not the case then the character set cannot be used in EDT.
In addition, EDT also requires a conversion property. It must be possible to convert all the occurring characters into `UTF16`.

Note

There is no guarantee that the glyphs of all Unicode characters are supported, for example the MT9750 V7 and Spool in OSD V6 do not contain the full Unicode scope but only the characters from the supported ISO 8859 variants 1,2,3,4,5,7,9,15.

In the text, the XHCS names for the names of the character sets and not the complete names, i.e. `EDF041` instead of EBCDIC.DF.04-1 or `UTF16` instead of UTF-16.

The `UTFE` character set is a BS2000-proprietary Unicode character set in which the characters are coded in byte sequences of variable length in the same way as in `UTF8`. The special feature of this character set is that not only all the characters from `EDF03IRV` but also all the relevant BS2000 control characters with the same code are coded in a single byte as previously. As a result, this character set is not just downwardly compatible with `EDF03IRV`, but also with the transport sequences used in communications with terminals.

The catalog entries of files and libraries in BS2000 may possess a character set specification. This specification is evaluated by the various products used in BS2000 such as OpenFT, SHOW-FILE and EDT.

Communication with a terminal is always performed in a character set. VTSU is responsible for this communication. However, VTSU makes it possible to specify a character set for each dialog step. Nevertheless, VTSU imposes certain restrictions as a function of the terminal mode. If the dialog step takes place in 7-bit mode then only `EDF03IRV` can be used In 8-bit mode, it is only possible to specify an EBCDIC character set which is compatible with an ISO character set variant that is supported by the terminal. If the terminal supports Unicode, then communication can take place in `UTFE`.

Example

The terminal is only able to depict ISO character set variant 1. In this case, `EDF041` or `EDF04DRV` can be specified as the character set in `VTSUCB`. The character set `ISO88591` cannot be specified since this is an ISO character set. The character set `EDF042` cannot be specified since it is not compatible with ISO character set variant 1.

For more information on XHCS and character sets, see [8].

3.4.2 Supported character sets

EDT only ever permits character sets which are supported by the current XHCS installation. This applies to both batch mode and interactive mode.
The communication character set must be compatible with the terminal. However, it can be different from the data character set.
The necessary conversions are handled via XHCS and the properties of the characters (uppercase/lowercase, special characters) are provided by XHCS.
By default, the Unicode character sets, all the EBCDIC character sets and the ISO character sets are permitted.

Handling invalid characters

Illegal byte sequences may occur in Unicode character sets. Thus, for example, in `UTFE` or `UTF8` several multibyte start characters may occur in sequence. EDT always rejects the entry of this type of illegal byte sequence both when reading files or variables and in the case of input in hex mode.

In `UTF16`, only characters from the surrogate range (`xD800–xDFFF`) are illegal.

All other characters, i.e. 2-byte sequences are accepted even if they cannot be depicted at the terminal. Special EDT semantic considerations are also ignored, e.g. the requirement that a character should not cause any line feed. In particular, even a *Byte Order Mark* (BOM) has no effect but is simply transferred as a character.

In the case of 7-bit character sets or incompletely defined 8-bit character sets all bytes are accepted and are taken over unchanged for reasons of compatibility. However, these undefined characters can never be converted into another character set.

Handling of national 7-bit character sets

In Unicode mode, all the 7-bit character sets that are defined in XHCS are permitted (for the handling of national 7-bit character sets, see section “The character set EDF03DRV” on page 58).

The @SHOW CCS statement can be used to query the currently supported character sets.

3.4.3 Strings

All strings are always interpreted and processed in a character set.

However, different character sets can be used. For example, data can be present in a file in one character set, be stored in a different character set in the work file and be displayed in a different character set again.
If necessary, in such cases, the string can be converted from the source into the target character set. Thus, for example, a file which is present in the character set EDF03IRV can be read into a work file in the character set UTF16 and displayed in UTFE at a Unicode terminal. It is then possible to insert any required characters (from the set of those available).
For logging purposes, the file can also be output to SYSLST using the character set EDF041.

The following table indicates how EDT determines the character set to be used in each case.

Source/target	**Character set for the string**
Input/output at a terminal	Communications character set
Reading from SYSDTA	Character set assigned to SYSDTA (provided by the BS2000 macro GCCSN). If *NONE, then EDF03IRV is used. If SYSDTA is assigned to a terminal then the communications character set is used.
Reading from a work file	Character set for the work file
Writing to a work file	Character set for the work file. If *NONE, then character set for the input data.
Reading from a string variable	Character set for the string variable.
Writing to a string variable	When a new string variable is created, character set in the CODE operand. If not specified, character set for the string. If no new string variable is created, character set of the string variable.
S variables or job variables	Character set from the CODE operand in the read/write statement. EDF041, if not specified.
Executing an @INPUT procedure	Character set of the file which contains the @INPUT procedure. If *NONE, then EDF03IRV is used.
Executing a @DO procedure	Character set of the work file which contains the @DO procedure.
Inserting from a DMS file or a library element	Character set from the file's catalog entry. If *NONE, then EDF03IRV is used.

Source/target	Character set for the string
Writing to a DMS file or a library element	If a new file is created, character set from the CODE operand or character set of the work file. On write-back, character set from the CODE operand or character set of the file or work file
Reading and writing from/to a POSIX file	Character set from the CODE operand in the read/write statement or the character set specified in @PAR CODE. By default EDF041.
Writing to SYSOUT	Character set assigned to SYSOUT (provided by the BS2000 macro GCCSN). If *NONE, then EDF03IRV is used. If SYSOUT is assigned to a terminal then the communications character set is used.
Writing to SYSLST	Character set assigned to SYSLST (provided by the BS2000 macro GCCSN). If *NONE, then EDF03IRV is used.

In the case of some statements (e.g. @CREATE, @SETJV), it is possible to specify multiple character sets which are initially joined in an intermediate result.
If all the strings involved have the same character set then this is also the character set of the intermediate result. If the involved strings have different character sets then the character set of the intermediate result is UTFE.
This intermediate result is then converted into the target character set.

3.4.4 Conversion and substitute characters

If necessary, EDT converts data from each permitted character set into another permitted character set. In such cases, if characters from the source data are not present in the target character set then a substitute character is used if the user has defined one with the statement @PAR SUBSTITUTION-CHARACTER. By default, no substitute character is defined (SUBSTITUTION-CHARACTER=*NONE).

When output is sent to a terminal in interactive mode, then, unlike in the case of other output destinations, the device-specific smudge character is used. If no substitute character is defined then a blank is used for all characters that are not present in the target character set when output is sent to SYSOUT/SYSLST. In all other cases, conversion is rejected, i.e. the associated statement is not executed.

The @CHECK statement (format 2) can be used to check whether a line range can be converted without loss into a target character set. This not only checks whether all the characters are present in the target character set but also that no length restriction will be exceeded. Strings may be significantly longer when converted into a Unicode character set.

Note

XHCS only converts compatible character sets with the result that conversion may pass via a Unicode character set (since these are always compatible).

3.4.5 Substitute character representation in Unicode

In Unicode mode, EDT permits the entry of characters which are, for example, not defined in the character set used in the source of the input, in the form of a substitute representation in which the UTF16 code of the character is specified directly. To do this, the @PAR ESCAPE-CHARACTER is used to declare a global or work file-specific escape character which initiates the substitute representation. By default, no substitution is performed (ESCAPE–CHARACTER=*NONE). The DATA–REPLACEMENT operand in the @PAR statement can be used to define the context in which substitution takes place.

By default, the substitute representation is only evaluated within statements and there only in literals (DATA–REPLACEMENT=OFF). Setting DATA–REPLACEMENT=ON causes this to be performed in data input as well.

The substitute representation has the form specUxxxx, i.e. the escape character is followed by a U or u (for Unicode) and exactly 4 hexadecimal numbers which specify the code of the character. If, for example, the escape character % has been specified using @PAR ESCAPE-CHARACTER='%', then the Greek Ω can be entered in the form %U03A9 or %u03a9 (the input is not case-sensitive).

If @PAR ESCAPE-CHARACTER=*NONE (default setting) has been declared either globally or for the current work file, if the substitute representation is formally incorrect or if no valid UTF16 character corresponds to the entered code then the substitute representation is treated as a normal string. The substitute representation is also not converted on data entry in F mode if the string for the substitute representation exceeds a column position for which a hardware tab has been defined.

If despite the specification of a valid UTF16 character, this cannot be converted into the target character set, then the procedure is the same as if an invalid character had been entered directly (e.g. via a corresponding keyboard).

The interpretation is independent of whether the characters entered using the substitute representation can be displayed at the screen or not. As described in the previous section, characters which cannot be displayed are depicted by means of the smudge character.

3.4.6 Communications character set

The communications character set is the character set used by EDT to exchange data with a terminal.

In Unicode mode, the communications character set can be different from the character set used in the current work file. Inputs and outputs are then converted accordingly if required.

Since VTSU is the interface to the terminal, the communications character set in interactive mode can only be a character set that is accepted by VTSU with the exception of EDF03DRV at a 7-bit terminal (see section "The character set EDF03DRV" on page 58). Restrictions consequently apply to the choice of the communications character set.

On start-up, EDT sets the character set from the terminal option `CODED-CHARACTER-SET` as the communications character set. The @CODENAME statement (format 2) can be used to set the communications character set explicitly. This setting remains valid until modified by a subsequent @CODENAME statement.

@CODENAME *AUTO, TERMINAL can be used to activate a mechanism by which EDT automatically attempts to select the most suitable character set. If it is communicating with a Unicode-compatible emulation, it specifies `UTFE` as the communications character set.

In the case of a terminal operating in 8-bit mode, i.e. if no communication via `UTFE` is possible, then the communications character set is implicitly defined by the character set of the work file displayed in the (top) work window.
If this character set changes, e.g. if EDT displays a different character set then the communications character set may also change. If the work file is empty and has the character set `*NONE` then, by default EDT takes the character set from the terminal option `CODED-CHARACTER-SET`. If 7-BIT is specified here, it uses `EDF03IRV`, otherwise the character set specified here. If the work file displayed in the (top) work window has a character set which is compatible with the terminal then this character set is specified.
If this character set is an ISO character set which is compatible with a character set which is itself compatible with the terminal then this is specified. If this is not the case, `EDF041` is used.

For communications with a terminal in 7-bit mode, the communications character set `EDF03IRV` is always used.

The same rules apply in L mode. This also applies when reading from `SYSDTA` (@EDIT ONLY), and `SYSDTA` is assigned to a terminal.

3.4.7 Character sets in work files

In Unicode mode, each work file may possess a separate character set.

Only in an empty work file is it possible for the character set to have the value *NONE. This is the setting after EDT has started and after the complete deletion of a work file using @DROP or @DELETE (format 2).

If the setting is *NONE then the character set can be defined either implicitly by inserting data in the work file, or explicitly by means of a @CODENAME statement. Once the character set for a work file has been set, all the characters which enter this work file are also converted into this character set.

If data is inserted into an empty work file with the character set *NONE then the work file is implicitly assigned the character set for this data on the basis of the source from which they arrive. For information on how EDT determines the character set, see the table in the section on strings.

If the work file already has a defined character set (i.e. not *NONE), then data is converted from its source character set when it is inserted in this work file. Independently of whether the work file is empty or not, the data is converted into the work file's character set on insertion.

If data is present in the work file then switching the character set with the @CODENAME statement causes this data to be converted into the new character set.

@CODENAME …,GLOBAL makes it possible to specify the same character set for all the work files using a single statement.

When migrations have to be performed, the @CODENAME statement provides additional performance scope. With @CODENAME …,FORCE=YES, it is possible to relabel the work file's character set. The work file is then assigned a new character set. However, the data it contains is not converted but remains unchanged. This option can be used to correct incorrectly labeled files. The FORCE operand is only applicable to 7-bit and 8-bit character sets.

3.4.8 Reading in files

In the following section, instead of talking about files **or** library elements, we refer only to **files**.

When reading in a file, EDT evaluates the character set of the catalog entry.

If the work file into which the file is to be read does not as yet have any character set (*NONE), the file's character set is defined for this work file and the file is read in. If the file has the character set *NONE, EDF03IRV is set.

If the work file already has a character set then the content of the file is converted from the catalog entry's into the work file's character set when it is read in. The work file does not have to be empty. It is therefore possible to read files with different character sets into a work file. The data is then present in the work file's character set. If the file contains characters which are not supported by the work file's character set then the file is not read in unless a substitute character has been defined. This is then used. If the file contains an invalid byte sequence (possible in Unicode character sets) then the file is not read in.

If the character set entered in the catalog is not supported then the file is also not read in.

If the file has been opened for real processing (see section "File processing" on page 131), then the work file must be empty and, in order to be defined, its character set must be the same as the file's character set or *NONE.

Example

```
   1.00 ·····························································
   2.00 ·····························································
   3.00 ·····························································
   4.00 ·····························································
   5.00 ·····························································
   6.00 ·····························································
```

```
CODENAME UTFE;COPY FILE=ADDRESSES·····························0000.00:00001(00)
```

To set the character set UTFE for work file 0, read in the file ADDRESSES with CCS=*NONE.

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
   2.00 DUCK     DONALD    WALTSTR.8      DISNEYLAND<·························
   3.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   4.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································************************
CREATE 0.01 'ÅNGSTRØM ANDERS    STERNWARTE     STOCKHOLM ';--··0001.00:00001(00)
```

Since the work file has the character set UTFE any characters from the Unicode character set can be entered in it provided that the terminal supports Unicode. This can be achieved as here, for example, by means of statements.

```
   0.01 ÅNGSTRØM ANDERS    STERNWARTE     STOCKHOLM<··························
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
   2.00 DUCK     DONALD    WALTSTR.8      DISNEYLAND<·························
   3.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   4.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································************************
····························************************0000.01:00001(00)
```

However, it can also be achieved by entering the data directly. Here **UE** has been replaced by **Ü** and **πλάτων** etc has been inserted in line 6.

```
   0.01 ÅNGSTRØM ANDERS    STERNWARTE     STOCKHOLM<··························
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MÜNCHEN<·····················
   2.00 DUCK     DONALD    WALTSTR.8      DISNEYLAND<·························
   3.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   4.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MÜNCHEN<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MÜNCHEN<·····················
   6.00 πλάτων             Academia      Athens<·····························
····························************************0000.01:00001(00)
```

3.4.9 Writing files

When new files are written, the character set of the work file is written to the catalog.

When the work file is written to a new file or an existing file with the character set *NONE, the character set specified with the new CODE operand or the character set of the work file becomes the character set for the file. If this character set is EDF03IRV and the file previously had the character set *NONE then the value *NONE is retained.

If, on a write operation to an existing file with a character set not equal to *NONE, the character sets of the file and work file are different (this occurs when a read-in file is written back after an explicit change of character set with @CODENAME or when an existing file is overwritten with new content), then an operand in the statements @CLOSE, @WRITE (format 1) and @XWRITE makes it possible to select the character set to be used for writing. If this operand is not specified then the write operation is rejected in batch mode. In interactive mode, the user is asked which character set is to be used.

If the file's character set is selected, EDT converts the work file before writing. If characters are present that cannot be depicted in the file's character set and if no substitute character has been defined (see @PAR SUBSTITUTION-CHARACTER), then the write operation is rejected with an error message. The user can then define the substitute character or modify the character set for writing and write the file again.

3.4.10 Copying between work files

The @COPY statement, @MOVE statement, @ON statement or statement codes can be used to copy data from one work file to another.

The source work file and target work file may have different character sets. In such cases, the data to be copied can be converted from the source into the target character set. If characters from the source data are not present in the target character set then the transfer is rejected unless the user has defined a substitute character.

3.4.11 Character set in a statement

The analysis of statements is always performed in UTFE.
The statements are converted from the character set in which they were read in into UTFE. This is always possible. This ensures, for example, that the '@' character can be used as the EDT statement symbol in all character sets.
Alongside the EDT statement symbol, EDT also makes it possible to redefine a number of other symbols of syntactic importance (single quotes and double quotes with @QUOTE, the line range symbol with @RANGE, wildcard symbols and filler characters with @SYMBOLS). To ensure consistent usage here, the set of permitted symbols is limited: Only the following symbols can be used:

!	"	#	$	%	&	'	(	)	*	+
,	-	.	/	:	;	<	=	>	?	@
[	\	]	^	_	`	{	|	}	~	

This means that only the special characters from EDF03IRV are permitted. Symbols which are present as special characters only in other character sets are rejected. in the case of EDF041, for example, these are § and ¤ as well as the symbols that cannot be entered at a normal keyboard such as ¼, ², ¶.

3.4.12 The character set EDF03DRV

The character set EDF03DRV is the only national 7-bit character set which is known in XHCS. For reasons of compatibility, it is handled in a special way if the terminal is recognized as a 7-bit terminal.

In this case, EDT also makes it possible to set EDF03DRV as the communications character set. However, the characters are also handled as EDF03IRV characters (without conversion) on transfer to and from the terminal. The correct display of the characters is ensured by the corresponding settings at the terminal or emulation. EDT is not able to check whether these settings are consistent. This is the user's responsibility.

3.4.13 String variables

String variables may be assigned any text such as a line in a work file. They can be accessed globally across work files. Every string variable has a content at all times since it is assigned the character '␣' (X'40') on initialization.

String variables always have a character set since they have a content at all times. Each string variable may have a different character set.

At EDT start time, string variables may be initialized by the S variables SYSEDT-S00..SYSEDT-S20. If one or more of these S variables are present then their values are taken over into the corresponding string variable. At this point, it is not possible to specify a character set. The string variables are therefore assigned the initial character set EDF041 irrespective of whether they were initialized with '␣' or by S variables.

The @CREATE statement can be used to assign a new value to a string variable. In this case, the CODE=name operand can be used to specify the resulting character set explicitly. If CODE is not specified then the resulting character set is the character set of the value that is to be assigned. If this value is obtained by joining strings with different character sets then the umbrella character set UTFE is used.

The character set for a string variable can also be defined using the @CODENAME statement. @CODENAME name,#S0 converts the content of #S0 to the corresponding character set. This character set is assigned to the string variable. @CODENAME name #S0,FORCE=YES can be used to relabel the character set (only permitted in the case of 7-bit and 8-bit character sets).

The @SET statement can also be used to assign a value to a string variable. If this value is specified as a string then a new string variable is created. It is then assigned the character set of this string variable.
If the value is obtained from the content of an integer variable, the content of a line number variable or line number or from the name of a printable string variable, then the string variable, if newly created, is assigned the character set EDF041. If only a part of the content is overwritten then the intermediate result is converted into the character set of the string variable and inserted.
If a binary value is inserted then the string variable is always newly created and assigned the character set EDF041.

3.4.14 S variables and job variables

The @GETVAR, @GETLIST and @GETJV statements can be used to transfer the contents of S variables or job variables to string variables or work files.

In Unicode mode, each string requires a character set specification. Since S variables and job variables record no information about their character sets, the character set must be defined at transfer time. This is possible using the new operand `CODE=name`. If the operand is not specified, the character set `EDF041` is used. If the statement is applied to a line in a work file then the content of the variable may be converted from this character set into the character set of the work file if necessary. If it is applied to a string variable then the value is taken over and the character set is assigned to the string variable.

Note

@GETVAR SYSEDT,CODE=name can be used to transfer the value of the `SYSEDT` variable to the string variable again while taking account of the character set.

The statements @SETVAR, @SETLIST and @SETJV can be used to generate S variables or job variables and assign a value to them.

In these statements, the `CODE=name` operand can be used to specify a character set into which the values are to be converted before assignment. If the operand is not specified, `EDF041` is used.

3.4.15 POSIX files

POSIX files also record no information about the associated character set.

In the statements used to read or write POSIX files, the operand `CODE=name` can be used to specify the character set in which the file's data is present or the character set to be used when writing the file. `name` may be any supported character set.

The specifications `CODE=ISO` - corresponding to `CODE=ISO88591` - and `CODE=EBCDIC` - corresponding to `CODE=EDF041` - are still possible for reasons of compatibility.

3.4.16 Outputs to SYSOUT and SYSLST

If `SYSOUT` is assigned to the terminal then output takes place in the communications character set and characters which cannot be displayed in this character set are replaced by the device-specific smudge character.

Otherwise, output to `SYSOUT` or `SYSLST` is performed in the relevant assigned character set which can be determined using the BS2000 macro `GCCSN`. If this character set is `*NONE` then `EDF03IRV` is used. If the output contains characters which cannot be displayed in the target character set and no substitute character has been defined then a blank is inserted.

3.5 EDT variables

The EDT variables are used to store values. These values can be integer values, strings or line numbers. In EDT procedures, these variables are used, for example, for the intermediate storage of values, as loop counters, for the input of strings (file names, search terms etc.) or to perform simple calculations.

EDT variables are only valid for the current EDT session, provided that the operating mode (see section "Introduction to the EDT operating modes" on page 21) does not change. They are globally visible, i.e. they can be set, used or queried from within all work files. If a value is assigned to a variable in a work file, then the variable is also available with the same value in other work files.

EDT provides three types of variables which can be assigned the following values. 21 variables of each variable type are available and are indexed from 0 to 20.

– Integer variables (#I0..#I20)
– String variables (#S0..#S20)
– Line number variables (#L0..#L20)

The EDT variables are assigned values by means of various formats of the @SET statement or using @CREATE (see @SET, format 1-5 and @CREATE). Another possibility is to supply the EDT variables with the content of job variables (@GETJV) or S variables (@GETVAR, @GETLIST) (see below).

The line number variable #L0 and the integer variables #I0 to #I3 should not be used since they may be overwritten with values if an @ON statement returns a hit.

Although job variables and S variables do not form part of the EDT variables, they are nevertheless discussed in the summary below since they are frequently used to store the content of EDT variables either between EDT sessions or permanently.

3.5.1 Integer variables

The integer variables (#I0..#I20) are used to store positive or negative integer values. The largest possible value is 2147483647 (2^{31} - 1).

The integer variables can be supplied with values by means of the @SET statement (Format 1) and with @GETVAR. @STATUS can be used to output the content of integer variables on screen. @IF (format 2) is used to evaluate the values of integer variables within EDT procedures.

When EDT is started, all the integer variables are preset to the value 0.

3.5.2 String variables

The string variables, (#S0..#S20) can be used to store strings in all character sets supported by EDT.
A string variable is similar to a record in a work file.
Like a record, it can accommodate a maximum of 32768 characters which can be addressed either individually or in sections thanks to column specifications, they can be searched in, replaced by other strings etc.

The integer variables can be supplied with values by means of the @SET statement, formats 2, 4 and 5, by means of @CREATE or using @GETJV, @GETVAR or @GETLIST. @PRINT can be used to output the content of the string variables on screen. @IF (format 2) is used to evaluate the values of string variables within EDT procedures and column specifications make it possible to address subsections or individual lines.

When EDT is started, string variables are preset to a blank in the character set EDF041 unless they have assumed the values of any S variables SYSEDT-S00..SYSEDT-S20 which may be present (see section "Starting EDT" on page 87).

Every string variable is assigned a character which specifies how the content of the variable is to be interpreted.
In the case of the @CREATE and @SET statements as well as in @GETJV and @GETVAR, this character set can be specified explicitly for the new string variable that is to be filled. If no character set is specified or if the string variables are already assigned values when EDT is started (see above) then the default values for the string variables are set as a function of the data source and system environment.

In statements in which the name of a string variable could be confused with a file name or a library element designation, it is necessary to prefix the variable name with a period in order to indicate that a string variable is intended, e.g. @OPEN FILE=.#S1 opens the file whose name is stored in the string variable #S1 whereas @OPEN FILE=#S1 opens the file with the name '#S1'.

3.5.3 Line number variables

Line numbers can be stored in the line number variables (#L0..#L20). The range of values is 0.0001 to 9999.9999.

The line number variables can be supplied with values by means of the @SET statement (format 3). @STATUS can be used to output the content of line number variables on screen. @IF (format 2) is used to evaluate the values of line number variables within EDT procedures.

When EDT is started, all the line number variables are preset to the (invalid) value 0.0000.

3.5.4 Job variables

In systems in which the JV subsystem (Job Variable Support) is installed, it is possible to use job variables in JV. Unlike the integer, string and line number variables, job variables persist after EDT has terminated and it is possible to access existing job variables in EDT.

In EDT, it is possible to delete job variable entries (@ERAJV), output the values of job variables or transfer these values to a work file or string variable (@GETJV), generate job variables and assign values to them (@SETJV) as well as output information about job variables or write this information to a work file (@STAJV). In EDT procedures, it is not possible to evaluate the content of job variables directly but instead only after this has been transferred to a work file or a string variable.

BS2000 does not assign any character set to job variables. When a job variable is read with @GETJV, it is therefore possible to explicitly define the character set in which EDT is to interpret the content of the job variable. If nothing is specified then the default mechanisms described in the section on character sets apply.

3.5.5 S variables

It is possible to access S variables in EDT (for S list variables, it is necessary to install SDF-P). Unlike the integer, string and line number variables, S variables persist after EDT has terminated and it is possible to access existing S variables in EDT.

In EDT, it is possible to output the content of S variables of type `STRING` and `INTEGER`, transfer this content to a work file, a string variable or an integer variable (@GETVAR), generate S variables of type `STRING` and `INTEGER` and assign values to them (@SETVAR) as well as to generate S list variables (type `LIST` with element type `STRING`), extend such variables, assign values to them (@SETLIST) and transfer their values to a work file (@GETLIST). In EDT procedures, it is not possible to evaluate the content of S variables directly but instead only after this has been transferred to a work file, a string variable or an integer variable.

BS2000 does not assign any character set to S variables. When an S variable is read with @GETVAR, it is therefore possible to explicitly define the character set in which EDT is to interpret the content of the S variable. If nothing is specified then the default mechanisms described in the section on character sets apply.

3.6 EDT procedures

EDT makes it possible to store sequences of statements in work files, cataloged files or library elements and execute these as required. Alongside statements, records may also be present which are then inserted in the current work file at the relevant current line number. Such sequences of statements and records are referred to under the umbrella term of EDT procedure.

EDT procedures are subdivided into @DO procedures and @INPUT procedures depending on their storage location and the statement used to start them.

@DO procedures

– are stored in an EDT work file (1..22),
– can only be executed as a whole,
– permit the passing of parameters, nesting, (conditional) branches and loops and
– are started with the EDT statement @DO.

@INPUT -procedures

– are stored in a file,
– can be executed as a whole or partially,
– do not permit the passing of parameters, nesting, branches or loops,
– can be started as an EDT start procedure when EDT starts or
– can be started with the EDT statement @INPUT.

The following sections start by explaining the concepts which apply to all procedure types. These form the basic rules for the creation and execution of EDT procedures. This is followed by a discussion of the possibilities offered by the @INPUT procedures and by integrating EDT procedures in BS2000 system procedures. The EDT start procedure is explained as a special type of @INPUT procedure. Finally, the language tools which can only be used in combination with @DO procedures are described, namely branches, loops and parameters.

3.6.1 Creating and executing EDT procedures

When creating and executing EDT procedures it is necessary to observe the following rules.

Reading statements and data lines

When executing an EDT procedure, EDT reads statements and data lines in L mode. If the EDT procedure is started in F mode, EDT switches to L mode, executes the procedure and then switches back to F mode.

The distinction between statements and data lines is therefore governed by the same rules as for input in L mode (see section “L mode” on page 126). In particular, lines with two consecutive (possibly separated only by blanks) EDT statement symbols or user statement symbols are interpreted as data lines. This makes it possible to construct other procedures dynamically within procedures and run these immediately (see example below).

Permitted statements

All the statements that are permitted in L mode may be used in EDT statements with the exception of the @DIALOG, @DROP and @INPUT statements (see section “L mode” on page 126).
In @DO procedures, the statements @GOTO, @DO (Format 2) and @PARAMS are also permitted. In @INPUT procedures, the @IF ... GOTO statement is tacitly ignored if the condition is not fulfilled and is rejected with the error message `EDT4942` if the condition is fulfilled.

Current and active work file, special work files

In L mode, the current work file is called with the @PROC or @SETF statement and in F mode with the @SETF statement or by entering the corresponding work window in the statement line. When EDT procedures are executed, the read statements and records always apply to the current work file. The current work file must therefore not be specified in a @DO statement.
The attempt is rejected with the message `EDT4906`.

Active work files are those work files which contain a @DO procedure which is currently being executed. If @DO procedures are nested then multiple work files can be active (up to 22), i.e. those that have been activated by a @DO and have not yet been exited with a @RETURN.

When @DO procedures are nested, an active work file may also be specified in @DO statements.

Recursive calls are therefore possible but again only to a maximum nesting depth of 23. An *active* work file must not be made into the *current* work file. A @PROC or @SETF statement for an active work file is therefore rejected with the message `EDT4959`.

@DO procedures can be stored and run in every work file with the exception of work file 0, i.e. in work files 1 to 22.
A @DO 0 is rejected with the message `EDT3209`.
Since many EDT statements write their output to work file 9 by default, it is advisable not to use this work file for @DO procedures.

EDT procedures and character sets

When an EDT procedure is executed, statements and records are read in the character set of the current EDT procedure.
In the case of @INPUT procedures, this is the character set of the cataloged file or library element; in the case of @DO procedures, it is the character set of that active work file which is being read. The read data records and the literals or text-type expressions may have to be converted if they refer to objects (e.g. the current work file or a string variable) to which another character set is assigned. For the precise rules, see section “Character sets” on page 47.

Example: Creating and calling a @DO procedure in F mode

```
  1.00 ·····························································
  2.00 ·····························································
  3.00 ·····························································
  4.00 ·····························································

 23.00 ·····························································
22·····························································0000.00:00001(00)
```

This switches to work file 22. The procedure is created in this work file.

```
  1.00 @COPY FILE=TESTFILE<·····································
  2.00 @PAR LOWER=ON<···········································
  3.00 ·····························································
  4.00 ·····························································
 23.00 ·····························································
0·····························0001.00:00001(22)
```

The statements are created in work file 22. After this, processing returns to work file 0.

```
  1.00 ·····························································
  2.00 ·····························································
 23.00 ·····························································
@do 22·····························0000.00:00001(00)
```

The procedure in work file 0 is called with @DO from work file 0.

```
  1.00 This is a test file which<·································
  2.00 has been read into work file 0<··························
  3.00 using a procedure.<·····································
  4.00 ·····························································
  5.00 ·····························································
 23.00 ·····························································
·····························0001.00:00001(00)
```

The result of running the procedure can be seen in work file 0.

Example: Calling a procedure in L mode as a @DO procedure

```
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 1:1-8
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) Processing switches to work file `1`.

(02) EDT statements are written to work file `1` (@DO procedure). Here, the L mode input mechanism is used to insert lines with two EDT statement symbols as data lines in the work file and truncate these before the second statement symbol (see section “L mode” on page 126).

(03) Processing returns to work file `0`.

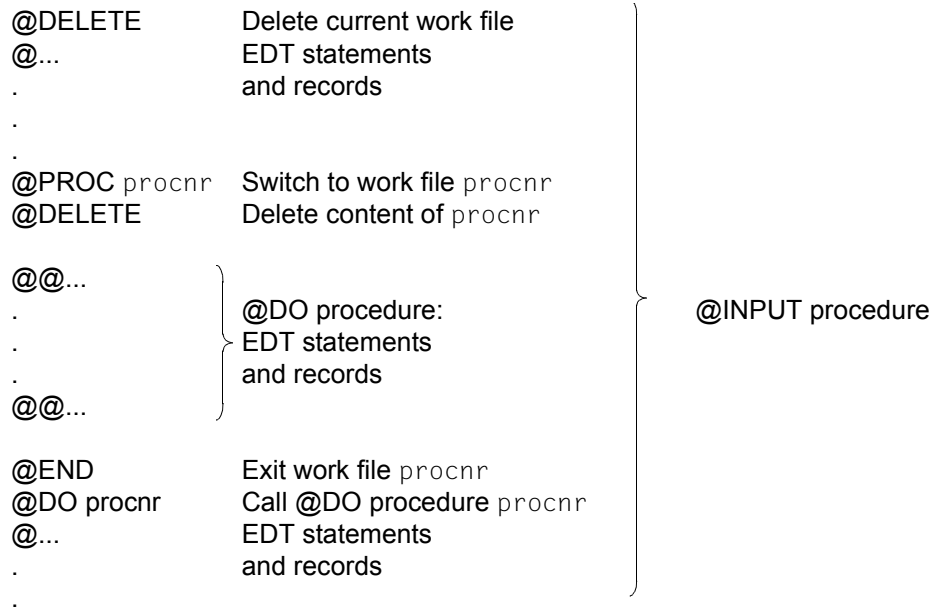
(04) Call of the @DO procedure. The statements in the work file are executed.

3.6.2 @INPUT procedures

EDT statements and records can be written as @INPUT procedures to a SAM, ISAM or. POSIX file or to a library element of a permitted type (see section “File processing” on page 131).

The advantages of an @INPUT procedure (permanent availability, informative procedure name) can be combined with the benefits of @DO procedures (nesting, branches and loops, parameter settings) by dynamically creating and executing one or more @DO procedures within the @INPUT procedure. This makes use of the possibilities of L mode input with two EDT application symbols.

Structure of a @DO procedure within an @INPUT procedure



Example

```
   1.00          @NOTE-----------------------------------------READ IN FILE NAME·
   2.00 @CREATE #SO0 READ 'FILENAME: '<·····································
   3.00          @NOTE---------------------------------OPEN WORK FILE 22·········
   4.00 @PROC 22<·····························································
   5.00 @DELETE<·····························································
   6.00          @NOTE------------------STORE FILE IN THE FORM YYMMDD IN #SO1···
   7.00 @ @SET    #SO1 = DATE ISO<·····································
   8.00 @ @SET    #SO1 = #SO1:1-8<·····································
   9.00 @ @ON     #SO1 CHANGE ALL '-' TO ''<···························
  10.00          @NOTE----------------STORE TIME IN THE FORM HHMMSS IN #SO2·····
  11.00 @ @SET    #SO2 = TIME<·····································
  12.00          @NOTE-----ASSEMBLE COPY FILE COMMAND AND STORE IN #SO3·········
  13.00 @ @CREATE #SO3 : '/COPY-FILE ',#SO0,',',#SO0,'.',#SO1,'.',#SO2<·········
  14.00          @NOTE-----------ISSUE COPY FILE COMMAND AS SYSTEM COMMAND·····
  15.00 @ @SYSTEM #SO3<·····································
  16.00          @NOTE---------------------------------CLOSE WORK FILE 22·····
  17.00 @END<·····························································
  18.00 @DO 22<·····························································
  19.00 ·····························································
  20.00 ·····························································
  21.00 ·····························································
  22.00 ·····························································
  23.00 ·····························································
@write file=backupcopy.input·····························0001.00:00001(00)
```

The procedure is created in F mode in the work file 0 and is stored as a SAM file under the name BACKUPCOPY.INPUT.

The @DO procedure consists of all the statements with two consecutive statement symbols separated only by blanks (@ @ - see below).

```
 20.00 ·····················································
 21.00 ·····················································
 22.00 ·····················································
% EDT0172 FILE 'BACKUPCOPY.INPUT' CREATED AND WRITTEN
@input file=backupcopy.input;22·····························0001.00:00001(00)
```

The procedure is called with @INPUT. The statements in the SAM file BACKUPCOPY.INPUT are processed. When this is done, the nested @DO procedure (lines 6.00 to 16.00, EDT-statements with more than one statement symbol @) are stored in work file 22 (see below) and executed immediately. Processing then switches to work file 22.

```
   FILENAME: testfile
```

When the statement @CREATE ... READ is processed, an input prompt is output. Once the file name TESTFILE is entered, a backup copy with the name TESTFILE.yymmdd.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 ·····················


·····························0001.00:00001(22)
```

The statements in the @INPUT procedure which have more than one application symbol have been stored in work file 22 in the form of a @DO procedure and the content before the second application symbol has been deleted.

3.6.3 Calling an EDT procedure in a BS2000 system procedure

An EDT procedure can be dynamically constructed and called within a BS2000 system procedure.

Example of an EDT procedure in a BS2000 system procedure

```
/SET-PROCEDURE-OPTIONS DATA-ESCAPE-CHAR=*STD
/BEGIN-PARAMETER-DECLARATION
/DECLARE-PARAMETER FILE,TYPE=STRING,INITIAL-VALUE=*PROMPT
/END-PARAMETER-DECLARATION
/SHOW-FILE-ATTRIBUT &FILE ------------------------------------------------(01)
/MODIFY-JOB-SWITCHES ON=(4,5) --------------------------------------------(02)
/START-EDT ---------------------------------------------------------------(03)
@PROC 20 -----------------------------------------------------------------(04)
@ @READ '&FILE'
@ @PAR LOWER=ON
@ @PAR SCALE=ON ----------------------------------------------------------(05)
@ @PAR INFORMATION=ON
@ @PAR EDIT FULL=ON
@END ---------------------------------------------------------------------(06)
@DO 20 -------------------------------------------------------------------(07)
@DIALOG ------------------------------------------------------------------(08)
@HALT --------------------------------------------------------------------(09)
/SET-JOB-STEP
/MODIFY-JOB-SWITCHES OFF=(4,5) -------------------------------------------(10)
```

(01) Check whether the file specified via the procedure parameter is present. If not, processing branches to SET-JOB-STEP.

(02) Set job switches 4 and 5 (see section “Job switches” on page 98).

(03) Call EDT.

(04) Processing switches to work file 20.

(05) The EDT statements in the @DO procedure are stored in work file 20.

(06) Processing returns to work file 0.

(07) Call the @DO procedure located in work file 20 (read in a file, distinguish between uppercase and lowercase, output a column counter, output an information line, set data window and mark column to overwritable).

(08) Switch to F mode dialog. After termination of interactive mode with @HALT or @RETURN, the system procedure run is continued at the point where it was interrupted.

(09) Terminate EDT

(10) Reset the job switches.

3.6.4 EDT start procedure

The EDT start procedure is a special @INPUT procedure which is run when EDT is started (see section "Starting EDT" on page 87). The EDT start procedure is determined on the basis of the following search hierarchy.

1. If the link name `$EDTPAR` has been assigned then the file associated with it is defined as the EDT start procedure and the search is terminated.
2. If a file named `EDTSTART` exists under the caller of EDT's user ID then this is used and the search is terminated.
3. If in the EDT installation, the system administrator has assigned the logical identification `SYSDAT.EDTSTART` to an existing, accessible file then this file is used and the search is terminated.
4. If the file `$.EDTSTART` exists under the default user ID and is accessible then this is used and the search is terminated.
5. If steps 1 to 4 fail to identify any file then no EDT start procedure is executed.

Each time EDT is called, `/SET-FILE-LINK` can therefore be used to set an individual EDT start procedure. In particular
`/SET-FILE-LINK FILE-NAME=*DUMMY,LINK-NAME=$EDTPAR`
can be set to prevent the execution of any EDT start procedure including the one set by the system administrator.

3.6.5 Unconditional and conditional branches

The @GOTO statement is used in @DO procedures to branch to a line. The line number is specified in the @GOTO statement. The line must exist and must not be located outside of the procedure.

If a @GOTO statement is specified in an @IF statement then a condition-dependent branch is possible in a @DO procedure. If the condition is fulfilled then processing branches to the line specified in the GOTO statement. If the condition is not fulfilled then the procedure continues with the statement which immediately follows the @IF statement.

To prevent the possible displacement of lines when the procedure is modified, the line numbers should be determined again with @SET, format 6 (abbreviated to @ for improved clarity) before branch destinations are specified (see also example).

Branches are not permitted in @INPUT procedures.

Example of unconditional and conditional branches

```
@PROC 2
@DELETE
@1.00 ------------------------------------------------------------------ (01)
@ @CONTINUE *** LINE NUMBER 1.00 AS OF HERE ***
@ @CREATE #S1 READ 'PLEASE ENTER SEARCH TERM: ' ---------------------- (02)
@ @ON & FIND #S1 MARK 5
@ @IF .TRUE. @GOTO 2 ------------------------------------------------- (03)
@ @CREATE #S2: 'NO HIT FOUND'
@ @PRINT #S2
@ @GOTO 3 ------------------------------------------------------------ (04)
@2.00
@ @CONTINUE *** LINE NUMBER 2.00 AS OF HERE ***
@ @DELETE MARK 5
@ @ON & PRINT #S1 ---------------------------------------------------- (05)
@3.00
@ @CONTINUE *** LINE NUMBER 3.00 AS OF HERE ***
@END
```

(01) Statement @1.00 sets line number 1.00 and, implicitly, the increment 0.01 in work file 2. The same applies equivalently for the other @SET statements.

(02) @CREATE...READ prompts the user to enter a search term. In the following line, all the lines which contain the search term are flagged with record mark 5.

(03) @IF checks whether there are any hits and, if there are, branches to line 2.00.

(04) If there are no hits, the corresponding message is output and processing branches to the end of the procedure.

(05) If there are hits, the record mark is deleted and the hit lines are output.

The lines with two consecutive statement symbols are read back into the lines defined by the @ statement in work file 2. The EDT procedure can then be executed with @DO 2.

3.6.6 External and internal loops

External loops make it possible to work through @DO procedures repeatedly in full. If only parts of a procedure are to be looped through repeatedly then it is necessary to use internal loops.

External loops are implemented by referencing a loop counter in the @DO procedure. The start value, end value and increment of this counter must be specified when the procedure is called in the @DO statement (see @DO statement page 285). The loop counter must be a special character and should not conflict with special characters which have a fixed meaning in EDT (e.g. `%` or `$`). In contrast, the characters `!` or `|` are suitable. Within the @DO procedure, the loop counter must be used like a line number (not like a line number variable, i.e. it cannot be modified in the procedure).

When the last statement in the @DO procedure is executed, the loop counter is increased or decreased by the specified increment and compared with the end value. If the comparison value is not greater or lower than the end value accordingly, the procedure is run again with the modified loop counter value. If the last statement in the @DO procedure is not executed, for example because the procedure has been exited with @RETURN, the procedure is not looped through again but is interrupted, perhaps before the specified end value is reached. It may therefore be necessary to write an artificial (empty) final statement (see @CONTINUE statement).

The special character representing the loop counter may be present, when the @DO statement is entered, in a character set other than that used in the called procedure. If necessary, it is converted in accordance with the same rules as the literals which may occur in other EDT statements (see section “Character sets” on page 47).

External loops can be replaced by internal loops. In an external loop, alongside the start and end value it is only possible to specify a fixed positive or negative increment. In an internal loop, it is possible to specify a variable increment, for example via a line number variable.

Example of an external loop

```
@PROC 3
@DELETE
@ @COLUMN 10 ON ! INSERT !:27-36:
@END
```

The above statement sequence constructs a @DO procedure containing the single statement @COLUMN... in work file 3.

If this @DO procedure is started with @DO `3, !=11,15`, then the values `11,12,13,14` and `15` are used sequentially for the loop counter `!` (the implicit increment is `1`). In these lines, the content of the relevant line (columns `27-36`) is inserted again at column `10`.

If this procedure is applied to a work file with a smaller increment (for example `0.1`) then some lines may be ignored. If all the lines are to be taken into account independently of the increment, an internal loop should be used (see example below).

Example of an internal loop

```
@PROC 4 ------------------------------------------------------------ (01)
@DELETE
@RESET
@1.00
@ @IF ERRORS : @GOTO 2 ---------------------------------------------- (02)
@ @IF #L10 > 15 @GOTO 2 --------------------------------------------- (03)
@ @COLUMN 10 ON #L10 INSERT #L10:27-36: ----------------------------- (04)
@ @SET #L10 = #L10 + 1L --------------------------------------------- (05)
@ @GOTO 1 ----------------------------------------------------------- (06)
@2.00
@ @CONTINUE
@END....

@SET #L10 = 11 ------------------------------------------------------ (07)
@DO 4
```

(01) A @DO procedure is constructed in work file 4.

(02) The procedure should be aborted if EDT errors occur.

(03) If loop counter `#L10` exceeds the value `15` then the procedure should be aborted.

(04) The content of the corresponding line in column `27-36` should be inserted again in the line defined via `#L10` in column `10`.

(05) The line number is set to the next existing line number. Specifying the value `1` instead of `1L` here would have the same effect as in the external loop (see above).

(06) Processing branches to the start of the loop.

(07) The start value of the loop counter is set outside of the @DO procedure and the procedure is then called.

Note

Line `11.00` must exist in the file that is to be processed and the last line in the file to be processed should be greater than `15.00`, otherwise the procedure is aborted with an error message.

3.6.7 Parameters

When @DO procedures are created in EDT, @PARAMS can be used to define formal parameters to which current values (current parameters) are assigned when the procedure is called with @DO.

The @PARAMS statement must be the first statement in a @DO procedure and may only occur once in the procedure. Both positional and keyword parameters are permitted. All the positional parameters must be defined before the keyword parameters.

A formal parameter starts with the character &. This is followed by a letter which in turn is followed by up to 6 letters or digits.

When the procedure is called, the parameters in the @DO statement are specified as the current parameters. It is also possible to set keyword parameters to a default value within the @PARAMS statement.

The default value is used if the corresponding keyword parameter is not specified in the @DO statement. When the procedure is called, the formal parameters in the procedure are replaced by the values of the current parameters or the default values.

The processing of these parameters should be considered as a two-stage process. First of all in the called procedure, the text of the formal parameters is replaced by the current parameters and the modified lines are then processed. Here, it may be necessary to take account of the presence of a number of different character sets, i.e. the character set in the statement (for the current parameters), the character set used in the work file that is to be run as a procedure and the character set of the current work file to which the statements in the procedure are applied or in which the records are inserted.

In the first stage of text substitution, it is therefore necessary to convert the character set of the statement into the character set of the executing procedure.
This applies both to the names of the specified current parameters and the names of the formal keyword parameters. If the current parameters contain a substitute representation for Unicode characters (see section “Substitute character representation in Unicode” on page 52), then they are not converted into the corresponding Unicode characters at the time of text substitution even if the current parameters are quoted.

In the second stage of processing (execution), both lines and literals in statements must be converted from the procedure's character set into the character set of the current work file. This operation includes the interpretation of a substitute representation for Unicode characters if substitute representation in a literal has been used.

It is not possible to pass through the current parameters unmodified since text substitution can take place at any position in a line in the executing procedure (in literals, in other operands or even in the statement name itself).

If in a Unicode environment, a procedure file is used which is present in a 7-bit or 8-bit character set, undesired character substitutions may therefore occur during the recoding of a parameter entered in Unicode into the character set used in the procedure file. This can only be prevented by using the substitute representation for Unicode characters (see above). For the precise rules governing recoding, see section "Character sets" on page 47.

Note

If EDT procedures with parameters are also to be used in BS2000 system procedures which also contain parameters, it is advisable to set the BS2000 parameter symbol to a value other than `&` (`/SET-PROC-OPT DATA-ESCAPE-CHAR=...`) in order to avoid conflicts.

Example for the use of parameters in an EDT procedure

In the following example, a file is read into work file 0. The records which contain the search term are copied into work file 5, prepared accordingly and output on the screen.

```
 1.     @PROC 4
 1.     @DELETE
 1.     @ @PARAMS &FILE,&SEARCH ---------------------------------------- (01)
 2.     @ @DELETE
 3.     @ @READ '&FILE'
 4.     @ @ON & FIND PATTERN '&SEARCH' COPY TO (5)
 5.     @ @PROC 5
 6.     @ @CREATE 0.01: '~' * 50
 7.     @ @CREATE 0.02: 'MENU ','&SEARCH'
 8.     @ @CREATE 0.03: '~' * 50
 9.     @ @RENUMBER
10.     @ @CREATE $+1: '~' * 50
11.     @ @PRINT
12.     @ @END
13.     @END
 1.     @DO 4 (MENU,CW 49) ------------------------------------------- (02)
 1.0000 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 2.0000 MENU CW 49
 3.0000 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 4.0000 CW 49 - 05.12. CORN FRITTERS, CHOCOLATE MOUSSE
 5.0000 CW 49 - 06.12. BEEFBURGER, BOILED POTATOES, YOGHURT
 6.0000 CW 49 - 07.12. CUMBERLAND SAUSAGE, FRENCH FRIES, FRUIT SALAD
 7.0000 CW 49 - 08.12. PENNE IN GARLIC AND MUSHROOM, TRIFLE
 8.0000 CW 49 - 09.12. COD IN BATTER, SAUTE POTATOES, APPLE FLAN
 9.0000 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
10.     @DO 4 (MENU,SAUSAGE) ---------------------------------- (02)
 1.0000 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 2.0000 MENU SAUSAGE
 3.0000 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 4.0000 CW 45 - 10.11. SAUSAGE AND EGG, FRENCH FRIES, CUSTARD PUDDING
 5.0000 CW 49 - 07.12. CUMBERLAND SAUSAGE, FRENCH FRIES, FRUIT SALAD
 6.0000 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

(01) Define the symbolic parameters (two positional parameters).

(02) Call the procedure with the associated current parameters. The formal parameters in the @READ, @ON statement are replaced by the current values on each @DO call.

3.7 Searching with @ON

There are ten formats of the @ON statement in which actions can be triggered depending on the search term. The search term defines one or more strings which can be searched for in a search range.

Alongside simple characters, the search term can include wildcards which act as place-holders for groups of characters. The wildcards stand either for precisely one character or for a string of any length. When a wildcard is interpreted, pattern matching occurs during the search operation.

The search term is bounded by a delimiter character on both the left and right. There are two different delimiters.

Depending on the delimiter character that is used, the start and/or end of the hit string is determined either simply by the hit string or by the hit string plus additional text delimiter characters before or after the search term. The two possible delimiter characters can be combined as desired when the search term is entered directly. The strings that are considered to be hits depend on the employed delimiter characters.

If wildcards are interpreted or if text delimiter characters are searched for in the search object then the search term and the hit strings may not be identical. In such cases, it is even possible that the individual hit strings will differ from one another.

The search term is only searched for in the line or column ranges which are specified in the @ON statement. The following sections discuss the details which are of relevance when searching using the @ON statement.

3.7.1 Case sensitivity

The @SEARCH-OPTION statement can be used to define whether a distinction should be made between uppercase/lowercase when searching for strings with the @ON statement. By default, searches are case-sensitive. If no distinction is to be made between uppercase/lowercase then the search term and search area are temporarily converted from lowercase to uppercase when the statement is executed.

The conversion is performed using the interface provided by `XHCS`. When lowercase/uppercase conversion is performed, the lengths of the strings do not change irrespective of the specified character set.

Example

@SEARCH-OPTION CASELESS-SEARCH=OFF @ON & C'suCH' TO 'SUCH'	The uppercase/lowercase notation of a character is taken into consideration during the search. This corresponds to the default setting of the @SEARCH-OPTION statement. Only the string ' `suCH` ' is converted into '`SUCH`'. Strings of the form '`such`' or '`sucH`' are not converted.
@SEARCH-OPTION CASELESS-SEARCH=ON @ON & C'suCH' TO 'SUCH'	Uppercase/lowercase notation is not taken into consideration during the search. All '`such`' strings in all possible case variations are converted into '`SUCH`'.

The `CASELESS-SEARCH` setting only applies to formats 1 to 3 and 5 to 10.
In format 4 it is not relevant since searches in this format are only conducted for marked lines.

3.7.2 Using wildcards in search terms

Alongside simple characters, it is also possible to specify placeholders for groups of character (so-called wildcards). There are two wildcards.

asterisk (Default value `*`) stands for a string of any length including an empty string. If specified more than once in succession then it is interpreted as a single asterisk, e.g. '`ABC**F`' is equivalent to '`ABC*F`'.

slash (Default value /) stands for precisely one character.

If the keyword `PATTERN` is specified then the wildcards are interpreted and pattern matching takes place. The wildcards are resolved into the shortest possible substring in the search range.

If the keyword `PATTERN` is not specified then the wildcards are handled as simple constant characters.

Example

@ON & PRINT 'AB*C'	Displays all the lines which contain precisely the string `AB*C`
@ON & PRINT PATTERN 'AB*C'	Displays all the lines which contain the strings `ABC`, `ABXC`, `ABCDEFG`, `ABXXXXXXC` etc.

Multiple wildcards may be present in every search term. A search term which consists only of wildcards is also permitted. The @SYMBOLS statement can be used to redefine the wildcards.

No wildcards can be specified within the substitute representation of Unicode characters.

3.7.3 Negative searches

If the keyword `NOT` is specified in the @ON statement then the records and/or string variables which do not contain the search term are identified. This is described in detail for the individual formats of the @ON statement.

Example

The work file contains the following lines:

```
1.ABCD
2.ABCE
3.ABDE
4.ACDE
```

With @ON & PRINT NOT 'AB' only the 4th line would be output.

With @ON & PRINT NOT 'ABC', the 3rd and 4th line would be output.

3.7.4 Delimiter characters

On input, the search term is bounded by a delimiter character on both the left and right. There are two different delimiter characters.

apostrophe (Default value ') specifies that text delimiter characters before or after the search term should not be searched for in the search range. The start and end of the hit string are therefore determined solely by the search term.

quotation mark
(Default value `"`) specifies that the hit string in the search range must be bounded by a text delimiter character before and/or after the search term. The start and/or end of the hit string are therefore defined by delimiter characters or by the first or last column of the column range to be searched.

The settings for the delimiter character *apostrophe* and for the delimiter character *quotation mark* can be modified using the @QUOTE statement.

The strings that are considered to be hits depend on the type of delimiter character used. If the search term is enclosed in *apostrophes* then an occurrence of the search term in the search range is considered to be a hit.
The wildcard *asterisk* has no significance if it occurs in the search term immediately next to the delimiter character *apostrophe*.

If the search term's left-hand delimiter character is the *quotation mark*, then for the search term to count as a hit it must be located either at the start of the line or a text delimiter character must be located immediately in front of it.

If the wildcard character *asterisk* is located immediately after the delimiter character *quotation mark* in the search term then the hit string extends to the next text delimiter character before the search term. If there is no text delimiter character then the hit string continues to the start of the line.

If the search term's right-hand delimiter character is the *quotation mark*, then for the search term to count as a hit it must be located either at the end of the line or a text delimiter character must be located immediately after it. If the wildcard character *asterisk* is located immediately before the delimiter character *quotation mark* in the search term then the hit string extends to the next text delimiter character after the search term.
If there is no text delimiter character then the hit string continues to the end of the line.

By default, EDT presets the set of text delimiter characters to the characters: blank (`X'40'`) and `+.!*();-/,?:'="`.
This character set can be redefined using the @DELIMIT statement.

To search for the delimiter characters *apostrophe* or *quotation mark* as part of a search term then these must be specified in duplicate.

The first and last columns of a specified column range act as text delimiter characters during the search in the same way as the start or end of a line.

The delimiter characters in a search term can be combined as desired.

Example 1

The work file contains the following lines:

```
1.ABCD
2.A,BCD
3.ABC,D
4.A,BC,D
```

With `@ON & PRINT 'BC'` 1 hit is identified in all 4 lines.
With `@ON & PRINT "BC'` 1 hit is identified in the 2nd and 4th lines.
With `@ON & PRINT 'BC"` 1 hit is identified in 3rd and 4th line.
With `@ON & PRINT "BC"` 1 hit is identified in the 4th line.

Example 2

The work file contains the following line:

```
1.XXX,ABCDEFGH-YYY
@ON 1 PRINT PATTERN 'EFG*"
```

The hit string is '`EFGH`' since it extends to the right as far as the next text delimiter character `'-'`.

```
@ON 1 PRINT PATTERN "*BCD'
```

The hit string is '`ABCD`' since it extends to the left as far as the next text delimiter character `'-'`.

Example 3

```
@ON & PRINT 'This ""string"" contains no ''X''.'
```

In this case, the search is conducted for the string

```
This "string" contains no 'X'.
```

3.7.5 Indirect specification of the search term

In the @ON statement, it is also possible to specify the search term via a line number, a line number variable or a string variable (in each case a column specification is possible). The line with the specified line number or string variable contains the search term.

Examples

```
@CREATE 6 : 'AB*C//D'
@ON 2-3 PRINT PATTERN 6:2-5:
```

The search term is therefore 'B*C/'.

```
@CREATE 1 : 'ABCDEFG'
@SET #L3 = 1
@ON 2-3 PRINT PATTERN #L3:4-7:
```

The search term is therefore 'DEFG'.

```
@SET #S0 = 'ABCD*E//F'
@ON & PRINT PATTERN #S0:3-8:
```

The search term is therefore 'CD*E//'.

In the case of an indirect specification, the search term is treated as if it were enclosed in *apostrophes*. The search for text delimiter characters in search ranges is therefore not possible in the case of indirect entry.

3.7.6 Search range

The @ON statement searches only in the specified search range.

Here, it is possible to specify one or more line ranges. A line range can also consist of just a single line. The line ranges are searched through in the specified order. It is also possible to specify a range of string variables for a line range.

An operand can be used to specify whether in each specified line range, the @ON statement identifies only the first hit line or searches through all the lines in the line range. By default, the @ON statement searches in all the lines in each specified line range.

Examples

```
@ON 1-3 FIND 'ABC'
```

In the @ON statement, only a single line range is searched.

```
@ON 1,2,3 FIND 'ABC'
```

The search range consists of three line ranges consisting of one line each. Depending on the operands, the search result may be different from in the preceding example.

```
@ON #S01,#S03.-#S04 CHANGE 'ABC' TO 'DEF'
```

In the @ON statement, the string variables #S01, #S03 and #S04 are searched for a hit.

A column range can be specified for a search in a line. It is not permissible to specify multiple column ranges. If no column range is specified then a default column range is used (see @SEARCH-OPTION).

Examples

```
@ON &:15-15: CHANGE 'A' TO 'D'
```

In the @ON statement, only a single column is therefore examined.

```
@ON &:15-25: PRINT 'XYZ'
```

In the @ON statement, only a contiguous column range is therefore searched.

3.7.7 Other search parameters

Depending on the operands, the search range can be searched through from either left to right or from right to left (R operand). By default, EDT searches the lines from left to right.

When a search is performed in a line, an operand can be entered to control as of how many occurrences a search term is considered to be a hit. By default, the first occurrence of the search term in a line is considered to be a hit.
Once the search term has been found, the @ON statement performs certain actions depending on the format in question.

Then, depending on the operand specification (ALL operand), the search for further hits in a line may be continued. In this case, the @ON statement takes account of the fact that the individual hit strings may not overlap.
This also applies if the hit strings are of variable length because the wildcard *asterisk* is interpreted or because a search is performed for text delimiter characters.
If the search is performed from left to right, the @ON statement therefore continues after the hit string. If the search is performed from right to left, the search for further hits is restricted to the columns located in front of the hit string.
As of the search for the second hit, every further occurrence of the search term in a line is considered to be a hit. By default, the @ON statement only identifies the first hit within a line.

3.7.8 Recording a hit

EDT records the results of the search for hits in local switches or in variables.

@IF (format 3) can be used to query whether a hit was identified the last time the @ON statement was run or whether the current work file was empty. If a hit was recorded, it is also possible to query the number of the column in which the first hit string starts.

In addition, EDT records the results of the search operation in accessible variables.

The number of the line in which EDT identified the first hit is recorded in line number variable #L0 and under the line number symbol '?'. If no hit is found or if the hit is identified in a string then the values of #L0 and '?' remain unchanged.

The number of the column in which the search term begins on the first identified hit (independently of whether this is a line or a string variable), is stored in the integer variable #I0 and the number of the column in which it ends in the integer variable #I1.
If no hit is found, the values of #I0 and #I1 remain unchanged.
This also applies to hits found in a string variable.
@PRINT #L0:#I0-#I1 can be used to output the hit string on screen if the hit was located in a line. If the operands V and ALL are specified then the number of hit lines or the number of string variables which contain the search term is stored in the integer variable #I2 and the total number of hits is stored in the integer variable #I3.
Independently of the specified character set, column specifications designate character and not byte addresses.

In the case of a negative search, following the occurrence of the first record in which the search term does not occur, the start position of the searched column range is recorded in integer variable #I0 and the end position in integer variable #I1.

If the end position of the column range is greater than the record length then the record length is stored in #I1.

Example

The work file contains a line with the string 'XXX-ABCD-YYY'.

Search term	Hit string	Content of #I0 and #I1
'ABC*Y“	ABCD-YYY	#I0 = 5; #I1 = 12
'ABC*Y'	ABCD-Y	#I0 = 5; #I1 = 10
"ABCD'	ABCD	#I0 = 5; #I1 = 8
'*BCD'	BCD	#I0 = 6; #I1 = 8

4 Using EDT

This section describes how EDT is integrated in the BS2000 system environment. This includes the underlying processes involved in starting, terminating, interrupting and monitoring EDT, an overview of the input and output flows and a presentation of the ways in which the system can be protected against undesired access via EDT.

4.1 Starting EDT

Since EDT as of V17.0A can be run in two operating modes (see section “Introduction to the EDT operating modes” on page 21), the EDT start commands have been extended accordingly.

When starting EDT it is possible to decide whether to opt for compatibility and do without the functional extensions – i.e. start EDT in compatibility mode – or to make use of the new functions and accept the presence of certain incompatibilities – i.e. start EDT in Unicode mode.

It is also possible to use EDT statements after start-up to switch between compatibility mode and Unicode mode.
It is always essential to start in Unicode mode if you need to make use of the functional extensions at the moment EDT is initialized, e.g. when running the EDT start procedure (see “EDT start procedure” on page 72).
For details of the functional extensions and incompatibilities in Unicode mode and the use of EDT statements to switch between modes, see section “Introduction to the EDT operating modes” on page 21 and chapter “Compatibility mode” on page 611.
The start process in Unicode mode is described below.

For information on starting EDT in compatibility mode, see section “Compatibility mode” on page 611.

4.1.1 The EDT start command

As of EDT V17.0A, EDT can be loaded and started in Unicode mode using the command /START-EDTU.
The /START-EDTU command makes it possible to select a specific EDT version if multiple versions coexist. /START-EDTU considers only EDT versions greater than or equal to V17.0A.
The /START-EDTU command may only be used under user IDs which have the necessary privileges (see section "Access protection" on page 99).

The alias for the /START-EDTU command is /EDTU.

START-EDTU	Alias: **EDTU**
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 =
Product version of EDT that is to be started.

VERSION = *STD
The version defined by the command /SELECT-PRODUCT-VERSION is selected. If there is no defined version, the system selects the highest possible version.

VERSION = <product-version 6..10> /
<product-version 4..8 without-correction-state> /
<product-version 3..7 without-manual-release>
Explicit specification of the product 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 of the JV software product [9]). For a detailed description, see section "Monitoring the EDT session with monitoring job variables" on page 96.

MONJV = *NONE
No job variable is used for monitoring.

CPU-LIMIT = *JOB-REST / <integer 1..32767>
The CPU time which EDT is allowed to use for execution. If this time is exceeded, then the system informs the user of this in interactive mode. In batch mode, the session is terminated.

CPU-LIMIT = *JOB-REST
If the operand CPU-LIMIT=STD has been specified in the /SET-LOGON-PARAMETERS command then the program is not subject to any time restriction.
If the operand CPU-LIMIT=t has been set in the /SET-LOGON-PARAMETERS command, the value defined during system generation is used as the time restriction for the EDT session.

PROGRAM-MODE =
Defines the addressing mode in which EDT is to run.

PROGRAM-MODE = *ANY
EDT is loaded in the upper address space and runs in 31-bit mode.

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

In interactive mode, EDT is started by default in F mode (full-screen mode, see section "F mode" on page 101), while in batch mode the default start mode is L mode (line mode, see section "L mode" on page 126).

If job switch 5 is set (see section "Job switches" on page 98) then L mode is also used for interactive mode. In this case, EDT uses RDATA to read input from SYSDTA.

EDT's mode of operation is also influenced by the declaration of the user's default character set with the /MODIFY-TERMINAL-OPTIONS command and by the character set defined for SYSDTA (for more information, see sections "Introduction to the EDT operating modes" on page 21 and "Character sets" on page 47).

The following initialization steps are executed during the EDT start phase:

1. Take over S variables defined via SDF-P into EDT string variables (see below)
2. Execute the EDT start procedure (see section "EDT start procedure" on page 72)

Processing takes place in the specified order. When EDT is started as a subroutine, these initialization steps are not performed.

When EDT is started, the string variables #S00..#S20 are initialized. If one or more of the S variables SYSEDT-S00..SYSEDT-S20 exist and are of STRING type then their content is assigned to the corresponding string variables. The content of S variables of other types is not taken over.

Since it is not possible to specify a character set at this point, the string variables are assigned the character set EDF041.

4.1.2 Calling EDT as a main program

For reasons of compatibility, it is still possible to call EDT using `/START-PROGRAM`. EDT is then loaded as a main program with one of the following BS2000 commands and is started in Unicode mode:

Command	AMODE
START-PROGRAM $.SYSPRG.EDT.170.EDTU	AMODE 31
START-PROGRAM *MODULE ($.SYSLNK.EDT.170,EDTCU, RUN-MODE=*ADVANCED)	AMODE 24

4.1.3 Calling EDT as a subroutine

EDT cannot only be called as a main program but can also be called as a subroutine by a user program.

The process of calling EDT as a subroutine is described in the manual "EDT Subroutine Interfaces" [1].

For a discussion of the specific considerations relating to the interaction between Unicode mode and EDT as a subroutine, see “Subroutine interfaces and operating modes” on page 616.

4.2 Interrupting and terminating an EDT session

The next two sections describe the special considerations to be borne in mind when interrupting or terminating EDT.

4.2.1 Interrupting an EDT session

In both F mode and L mode, the EDT session can be interrupted by means of @SYSTEM or by pressing K2. In both cases EDT remains loaded.

If, during the period of interruption, other programs are loaded via the BS2000 command interface (e.g. with /START-PROGRAM or /LOAD-PROGRAM) or if procedures are started which load other programs then EDT is unloaded without any query being issued and it is then not possible to continue the EDT session.

It is possible to return to the interrupted EDT work mode using the /RESUME-PROGRAM command. The /RESUME-PROGRAM command continues the EDT session at the point where it was interrupted.

In F mode, if the work window in which the EDT session was interrupted is not displayed or is incomplete following /RESUME-PROGRAM, then the original content can be restored by pressing K3. If EDT is interrupted using K2 then all input which has not yet been transferred is lost.

It is also possible to continue an interrupted EDT session mode using the /INFORM-PROGRAM command. When this is done, any message passed in the command is ignored.

If the @SYSTEM statement was specified in a statement sequence in F mode or in an input block in L mode (BLOCK mode) and /INFORM-PROGRAM is used to return to EDT after the interruption then a message is output and the remainder of the statement line or input block after @SYSTEM is not executed.

If, at the time of interruption, EDT had not yet fully processed the lines in a @DO or @INPUT procedure then the interrupted processing is aborted following a return with /INFORM-PROGRAM. A message is output and the remaining lines are not executed.

If a line range is being executed in an EDT statement at the time of interruption then execution of the statement is usually aborted and a message output on return to EDT with /INFORM-PROGRAM.

Note

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 “Access protection” on page 99).

4.2.2 Terminating an EDT session

The statements @HALT, @RETURN (outside of procedures), @EXEC and @LOAD and, in F mode, the K1 key terminate EDT normally. When this is done, EDT closes all open files.

In interactive mode, it may also be possible to terminate EDT with @END. In L mode, the messages

```
%  EDT4939 '@END' WITHOUT '@PROC' STATEMENT
%  EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)?
```

are first output to prevent any unintentional termination of EDT.

If EDT is running a screen dialog (after @DIALOG), then the statements @HALT, @RETURN (outside of procedures), @END and the K1 key terminate the screen dialog but not EDT itself. In this case, the user is not asked to confirm.

@HALT ABNORMAL can be used to force an abnormal termination of the EDT session if EDT was started as a main program. If EDT was started as a subroutine, @HALT ABNORMAL returns control to the calling program and issues a special return code.
If there are any unsaved work files when termination is requested, EDT is not immediately terminated if it is running in interactive mode. After the message

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

the numbers of the work files containing unsaved data are output. The user then sees the following query:

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

If the user replies N, the EDT session continues and the user can resume work, for example by writing back as yet unsaved files. If the user replies Y the unsaved work files are lost and EDT is terminated.

In F mode, if EDT is terminated with the K1 key then the confirmation query

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

is output in the work window's message line even if there are no open work files.

When EDT is terminated, the content of the string variables #S00..#S20 is assigned to the corresponding S variables SYSEDT-S00..SYSEDT-S20 if these exist and if they are able to accept a value of type STRING. The values of the string variables are passed in the character set EDF041. If conversion errors occur, the value is not transferred. If the value is more than 4096 bytes long then it is truncated and only the first 4096 bytes are transferred. No messages are output. This assignment step is omitted if it has already been performed manually using @SETVAR with the KEEP operand or if EDT was started as a subroutine.

If the event *Program runtime exceeded* occurs (EDT runtime exceeds the value specified for CPU-LIMIT in the /START-PROGRAM command), then a message is output to SYSOUT and EDT is terminated abnormally if it is running in batch mode.

If the interrupt event PROCHK (program error) or ERROR (unrecoverable program error) occurs and the EDT data area is still addressable then message EDT8910 is output and specifies the program counter and interrupt weight. A memory dump is generated and EDT is terminated abnormally.

To control system procedures in which EDT is called, information about the cause of EDT termination and about the EDT session is provided both in the event of normal termination with @HALT, @RETURN (in interactive mode, also with @END), or abnormal termination brought about by the system or by the user with @HALT ABNORMAL.

This information is not available if the EDT session is aborted with the @EXEC or @LOAD statements or with the BS2000 /CANCEL-PROGRAM command or if EDT is unloaded by means of another BS2000 command.

4.2.3 EDT command return code

EDT supplies a command return code that can be used by SDF-P for the control of S procedures. The command return code makes it possible to react specifically to certain error situations.

The command return code consists of three parts:

– the main code which corresponds to a message code by means of which more detailed information can be queried using the command `HELP-MSG-INFORMATION`
– subcode1 (`SC1`) which assigns the error situation that has occurred to an error class which makes it possible to estimate the severity of the error
– 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. Only messages of message severity 0, 1, 2 were issued (information, warnings, no syntax errors)
5	0	EDT8000	Normal termination of the EDT session. At least one message of message severity 4 or 5 occurred (function or execution errors, no syntax errors)
10	0	EDT8000	Normal termination of the EDT session. At least one message of level 3 severity was issued (syntax error in a statement)
50	64	EDT8101	Abnormal termination by the user (@HALT ABNORMAL)
100	64	EDT8200	Abort due to time overrun (program runtime exceeded)
100	64	EDT8292	Read error. Program aborted.
100	64	EDT8293	Write error. Program aborted.
150	64	EDT8910	Program interruption. Abnormal abort with memory dump
150	64	EDT8001	Abnormal termination after program interruption
200	64	EDT8002	Error in dynamic loading of EDT mode.
200	64	EDT8003	Insufficient virtual memory available
200	64	EDT8005	EDT initialization error
200	64	EDT8006	Installation error

For information on the messages and message levels, see section “Message texts” on page 638.

If EDT is terminated abnormally, it is possible to query the components of the return code using the SDF-P functions `SUBCODE1()`, `SUBCODE2()` and `MAINCODE()`.

If EDT is terminated normally, the /SAVE-RETURNCODE command can be used to save the return code for evaluation (for more detailed information on command return codes and for querying return codes, see the SDF-P User Guide [7]).

Example for the querying of return codes

```
/MODIFY-JOB-SWITCHES ON=5
/START-EDT
@LOG NONE
@...
@DIALOG
@...
@HALT
/SAVE-RETURNCODE
/IF-BLOCK-ERROR
/    WRITE-TEXT 'ERROR: &SUBCODE1, &SUBCODE2, &MAINCODE'
/ELSE
/    WRITE-TEXT 'EDT TERMINATED NORMALLY'
/    IF (&SUBCODE2 > 5)
/        WRITE-TEXT 'A SYNTAX ERROR HAS OCCURRED'
/        RAISE-ERROR MAINCODE=EDT3002
/    END-IF
/    ...
/END-IF
/HELP-MSG-INFORMATION &MAINCODE
/MODIFY-JOB-SWITCHES OFF=5
```

4.3 Monitoring the EDT session with monitoring job variables

EDT execution can be monitored with a BS2000 job variable.

```
/START-EDTU MONJV=jvname
```

If a monitoring job variable is specified when EDT is started then this is filled with a value when EDT terminates.

The value of the job variable consists of

– a 3-byte status display,

– a 4-byte return code display,

The following table indicates the values which EDT may assign to 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 with DUMP Initialization error	Abnormal		2100 2150 3200	

The value and meaning of the last 3 digits of the return code correspond to subcode 2 (SC2) of the command return code.

4.4 Input and output

Input to EDT may take the following form

– from the screen (primary source)
– from the system file SYSDTA
– from a SAM or ISAM file
– from a library element
– from a POSIX file
– from another EDT work file (@DO procedure)

When reading input, EDT distinguishes between data (texts) and statements.

Whether EDT reads from the terminal or from SYSDTA depends, on the one hand, on the system environment (dialog or batch mode) and, on the other, on the EDT work mode (F mode or L mode) (see section “EDT work modes” on page 101). The work modes can, in turn, be set by means of job switches (see section “Job switches” on page 98) and statements (@EDIT statement).

The other input sources must be explicitly declared in a statement (see the statements @OPEN, @COPY, @READ, @GET, @XOPEN, @XCOPY, @INPUT, @DO)

Output from the EDT may take the following form:

– to the screen (primary target)
– to the system file SYSOUT
– to a SAM or ISAM file
– to a library element
– to a POSIX file
– to the system file SYSLST

Whether EDT writes to the terminal or to SYSOUT depends, on the one hand, on the system environment (dialog or batch mode) and, on the other, on the EDT work mode. The work modes can, in turn, be set by means of job switches (see section “Job switches” on page 98) and statements (@EDIT statement).

The other output targets are declared either directly or indirectly in a statement (see the statements @CLOSE, @WRITE, @SAVE, @COPY, @XCLOSE, @XWRITE, @PRINT, @LIST)

For details, see chapter “File processing” on page 131.

4.5 Job switches

There are 5 job switches whose settings are evaluated by EDT for runtime control purposes. Before the EDT session, the switches can be set or reset using the system command /MODIFY-JOB-SWITCHES. During the EDT session, this is possible using @SETSW.

4.5.1 Job switch 4

If job switch 4 is set before EDT is loaded then, after loading, the messages of the dynamic binder loader (BLS05xx), the EDT start message in L mode (EDT0001) and, on EDT termination, the message

```
%  EDT8000 EDT TERMINATED
```

are not output. The following messages are also not issued:

```
%  EDT0900 EDITED FILE(S) NOT SAVED!
%  EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)?
```

If job switch 4 is set before EDT is loaded in batch mode then @LOG NONE is set, i.e. no logging is performed during the EDT session.

4.5.2 Job switch 5

If job switch 5 is set when EDT starts then L mode is used. EDT uses RDATA to read input from SYSDTA. The same effect (reading from SYSDTA with RDATA) can be achieved by entering @EDIT ONLY at the screen. Instead of the current line number, EDT outputs * as a prompt in interactive mode.

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

An explicit switchover (with @EDIT without operands in L mode or with @EDIT FULL SCREEN in F mode) is nevertheless still possible.

4.5.3 Job switch 6

If output is sent to SYSLST (e.g. @LIST statement), EDT normally writes no more than 132 characters per line. If job switch 6 is set then EDT writes up to 160 characters per line. Longer outputs are distributed over multiple lines.

Job switch 6 must already be set when EDT is started.

4.5.4 Job switch 7

This job switch can be set either before EDT starts or during the EDT session. It prevents EDT from automatically releasing previously assigned disk space after writing SAM or ISAM files. Normally, EDT releases unoccupied disk space using the FILE macro (see chapter "File processing" on page 131).

4.5.5 Job switch 8

In batch mode, EDT writes messages and outputs a series of statements (e.g.: @STATUS) to SYSLST. If job switch 8 is set then EDT writes this output to SYSOUT (see section "System files" on page 149).

Job switch 8 must already be set when EDT is started.

4.6 Access protection

There are two ways of protecting the system against unauthorized access via EDT.

– EDT may only be started if the user ID possesses a specific privilege.
– Protection by means of uninterruptible BS2000 system procedures which check which EDT statements are called

4.6.1 Constraints for privileged user IDs

The /START-EDTU command can be entered under all user IDs which possess the privilege TSOS and/or STANDARD-PROCESSING. If a user ID has only one or more of the following privileges, then EDT is started but any statements with security implications are rejected.

Privilege	Meaning	System ID
HARDWARE-MAINTENANCE	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

The following statements have security implications for user IDs with these privileges:

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

If used with the above-mentioned user IDs, these statements are rejected with error message EDT4976.

4.6.2 Uninterruptible procedures

If BS2000 system procedures are protected against interruption by the caller by means of INTERRUPT-ALLOWED=NO then the following applies to EDT:

– It is not possible to switch to system mode by means of K2.

– If EDT procedures are aborted by means of K2 then EDT issues the message EDT0913 to ask whether any actions are to be performed.

– In interactive mode and when input is read from a file (read with RDATA from SYSDTA, processing of an EDT start procedure), the statements with security implications – @SYSTEM, @EXEC, @RUN, @LOAD, @UNLOAD and @USE – are rejected unless the statements are issued by the protected procedure itself (SYSDTA=SYSCMD).

5 EDT work modes

EDT offers two work modes for processing data:

– In FULL-SCREEN mode (F mode) the whole screen is available in 23 work files (0–22) 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 available for the entry of data and statements at any time.
To make it possible to distinguish between records and statements, statements must start with the statement symbol (by default: @) or with the user statement symbol (see @USE statement, page 547).

5.1 F mode

In F mode, EDT provides screen-oriented file processing for SAM and ISAM files, library elements and POSIX files. A total of 23 work files (0–22) are available to the user.

Screen-oriented means that in the data area that is displayed on the screen,

– the data can be overwritten in any order
– text can be deleted and inserted anywhere in a screen line
– text can be entered at the end of the file or in newly inserted screen lines

In addition to the possibility of making changes directly at the screen, users can control file processing by means of:

– statements entered in the statement line
– statement codes entered in the statement code column
– statements entered in the data window (e.g. division of lines)
– record mark
– function keys

The formatted screen output is referred to as the work window. This displays the data of the work file which has been written to this file by means of screen input or by reading SAM or ISAM files, library elements or POSIX files.

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

Supported terminals

In EDT's F mode, the characteristics of the employed terminal are clearly of special importance. EDT was designed for the 8160 and 9750 terminals and upwardly compatible devices as well as for the corresponding terminal emulations and the associated characteristics.

It is usually only possible to work purposefully with Unicode files if using a terminal emulation which permits the input of Unicode characters and which is also able to display these characters correctly in the screen window (e.g. MT9750 as of V7.0 with terminal type 9763 and Unicode terminal mode). The 3270 terminal is no longer supported in the EDT Unicode mode. It can, however, still be used in compatibility mode.

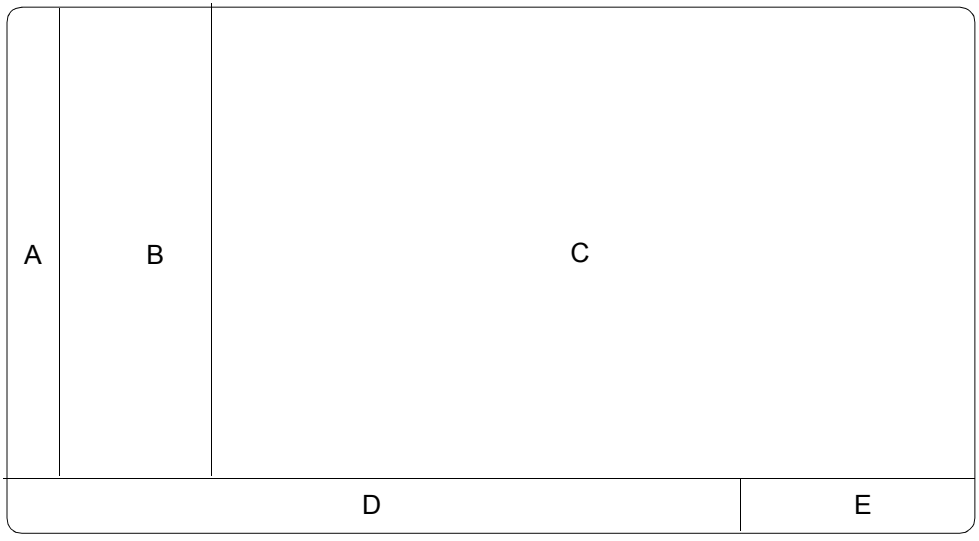
During data transfer with a terminal, it is only ever possible to use one character set per dialog step. A dialog step with a different character set modifies the terminal display globally and not just the presentation of the most recently transferred data. This results in the screen being deleted and then reconstructed.

Since EDT permits the simultaneous processing of work files which are coded in different character sets, it must be possible to define the character set used for communication with the terminal independently of the currently visible work files (in particular if the terminal does not support the work file's character set). This is done using the @CODENAME statement (format 2).

This statement simply modifies the screen display and possibly also the interpretation of the screen input. It does not modify the character set used in the work files or the character set of the underlying DMS file “Character sets” on page 47).

5.1.1 The work window

The work window subdivides the screen into fields with different functions. The diagram below indicates the structure of the work window with line number display active.



A = Statement code column
A + B = Line number display
C = Data window
D = Statement line
E = Status indicator

The display at the terminal is always constructed using the communications character set specified for the terminal. This can be selected automatically by EDT or set explicitly by the user with the @CODENAME statement (format 2) (see also section “Communications character set” on page 53).

A valid character set that has been explicitly specified using @CODENAME name, TERMINAL is always set independently of the content of the work files and of the character sets defined for these work files. This character set remains valid until the user changes it again or reactivates EDT's automatic character set selection capability with @CODENAME *AUTO, TERMINAL.
When EDT starts, the character set declared by means of `/MODIFY-TERMINAL-OPTIONS` is specified.

If EDT's automatic character set selection capability for communications with the terminal is active then EDT proceeds as follows:

– If the terminal supports the display of Unicode character sets then the character set UTFE is defined as the communications character set even if it differs from the character set declared in /MODIFY-TERMINAL-OPTIONS.

– If the terminal supports 8-bit character sets (but not Unicode), the character set declared by means of /MODIFY-TERMINAL-OPTIONS is defined. If 7-BIT is specified, EDF03IRV is used as the communications character set, otherwise the specified character set is used.

– If the terminal can only operate in 7-bit mode, EDF03IRV is used.

In certain situations, EDT's automatic character set selection capability results in the communications character set being changed in order to optimally adapt the depiction to the displayed contents. A switchover may occur whenever the current work file is changed or when the EDT automatic character set selection capability for communications with terminals is activated (switched on).

– If the terminal supports the display of Unicode character sets then UTFE is set immediately (even without a change of work file).

– If the terminal supports the display of Unicode character sets then the definition of the communications character set is not modified when the work file is changed. If the terminal can only operate in 7-bit mode, the communications character set never changes.

– If the terminal supports 8-bit character sets (but not Unicode) then the defined character set is the one declared for the work file displayed in the (topmost) work file provided that this character set is supported by the terminal.
If this character set is an EBCDIC or Unicode character set and is not supported by the employed terminal then EDF041 is set. If it is an ISO character set then the data is converted to EBCDIC in a way which is transparent to the user and the EBCDIC character set which is assigned to the ISO character set is used.
If the work file displayed in the (topmost) work window is empty and has the character set *NONE then the character set declared by means of /MODIFY-TERMINAL-OPTIONS is used.

The character set specified for output also defines the character set in which input at the terminal arrives at EDT.
Independently of this character set which is used for transfer purposes, it may nevertheless be necessary to reinterpret the input depending on whether or not it is to be evaluated globally (statement codes, file names in statements etc.) or refers to objects with a separate character set (work files, string variables etc.). The rules for this interpretation of the input, in particular in the case of literals, are described in section “Character sets” on page 47.

5.1.1.1 Statement code column

Functions can be triggered by entering single-character statement codes in the statement code column.

In the default setting, when records are displayed in the data window, the statement code column can be overwritten while the data window is protected against overwriting. The data window lines are set to overwritable only when statement codes are entered in the statement code column or when data is transferred with F2. It is not then possible to specify any statement codes in the overwritable screen lines.

As an alternative to the default setting, the statement @PAR EDIT-FULL=ON can be used while the line number display is active (@PAR INDEX=ON) to set the data window and the statement code column to overwritable. It is then possible to enter a statement code and at the same time modify data in this screen line (see @PAR EDIT-FULL).

Invalid entries in the statement code column can be deleted by overwriting them with blanks or NULL characters.

5.1.1.2 Line number display

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

With the exception of the start of the line number display which is also the statement code column, it is not possible to overwrite the line number display.

Line numbers are displayed in 6-digit form. Four of these digits precede the decimal point and two follow it. A non-overwritable blank separates the line number from the data line.

The complete line number with its total of four digits after the decimal point is displayed in L mode.

5.1.1.3 Data window

In the data window, the current work file is displayed on the screen. A work file consists of records. These records are output in the data window's screen lines and a record may be longer than a single screen line. In this case only part of the record is visible in the data window. The data window represents a section of the work file. However, it can be moved to a new position in the work file.

Provided that their length does not exceed the number of characters that can be displayed at the terminal, records that are longer than a data window line can be displayed in full 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 (by default the NULL character) and are set to overwritable. These lines are already sequentially numbered with the set default standard increment. The same depiction is also used if the data window is positioned so close to the end of the file that there are fewer records to be displayed than there are lines present in the data window.

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

By default, the records in the data window cannot be overwritten. Before they can be modified, individual records must be set to overwritable using the statement codes X or E or all the data window lines must be set to overwritable with F2. In EDIT-FULL mode, which is set with @PAR EDIT-FULL=ON, all the records in the data window are overwritable at all times. In EDIT-FULL mode, statement codes can be entered in the statement code column at the same time as entries for the same line are made in the data window (see @PAR EDIT-FULL).

The function keys F1 to F22 as well as the DUE and DUE2 keys can be used to transfer input to the terminal. The keys K1 to K15 do not transfer the input and any entered text is lost. Some function keys also trigger special actions in EDT, for more information see sections "Function keys in F mode" on page 123 and "Function keys in L mode" on page 128.

Empty lines and new lines

The files that are to be processed by EDT may contain records of length 0. In the case of POSIX or SAM files, the records genuinely have length 0. In the case of ISAM files with standard attributes, the records may have the length 8 or 16 (in the case of files coded in UTF16).

To permit the depiction of records of length 0 in the data window, EDT in Unicode mode indicates the end of the record using a terminal-specific character LZE (Logical Line End). The terminal fills the remainder of the screen to the right of LZE with protected NULL characters (X'00'). If the end of the record is located outside of the data window then LZE is not depicted and the screen line then ends with the last character of the record that is still visible or consists only of NULL characters. A record of zero length is therefore depicted in the data window by a screen line which consists only of the character LZE in column 1 and protected NULL characters (empty line). If LZE is entered in column 1 of a screen line then a record of length 0 is created for this line in the work file.

Empty lines should be distinguished from *new lines* which EDT provides in F mode after the last record in the file or during the processing of the statement codes 1..9 or I. These screen lines do not (yet) correspond to any record in the work file and consist only of NULL characters (X'00') without LZE (and can be overwritten).

The LZE character can usually be omitted during input. It only has to be entered when the record is intended to end with NULL characters. EDT ignores NULL characters at the end of the entered screen line, i.e. all the NULL characters up to the first character which is not NULL (this can be an LZE or another character) are truncated from the right. The LZE itself is not taken over into the record. Since new lines only consist of NULL characters they are ignored overall on input and are not inserted in the work file. In contrast, entering an LZE in column n of a new line would cause a record with n−1 filler characters (default value: blank) to be inserted in the work file after data transfer. In particular, a record of length 0 would be inserted for n=1.

The terminal does not permit any entries to the right of the LZE character in a screen line When adding entries to a line, it is therefore necessary to activate the terminal insertion mode or to overwrite the LZE character.

Treatment of filler characters in the data window

Since the LZE deletes the remainder of the screen line at the terminal, it is not possible to display any characters in the remainder of the line apart from the character specified for the terminal at the hardware level (normally NULL). The EDT statement @SYMBOLS FILLER can therefore no longer be used in Unicode mode to define the filler character displayed between the end of the record and the end of the screen line in F mode. However, for reasons of compatibility, the filler character specified in this way (default value: NULL) is still replaced by a blank on entry within a record. If all the characters entered in a record, i.e. including the NULL characters, are to be taken over unchanged into the work file, the @SYMBOLS FILLER='␣' statement must be used to change the filler character to blanks.

Treatment of NULL characters in the data window

If a screen line only contains NULL characters, i.e. no LZE, and if the displayed section comprises the entire record then the screen line is not taken over into the work file or is even deleted from it when it existed before.

When a record is entered (typed into an empty file, appended at the end of a file, inserted in screen lines which are provided for insertion after one of the statement codes 1..9 or I), NULL before or between other characters (including before LZE) are converted into blanks unless @SYMBOLS FILLER has been used to specify a character other than NULL as the filler character (see above).

NULL characters at the end of a screen line are ignored. In the case of records which are longer than the section that can be displayed on the screen, NULL characters at the end of the screen line cause the remaining text to be placed at the first location that is not equal to NULL, thus shortening the record. This mechanism is particularly common when performing insertions with the statement code E.

The statement code D should always be used to delete an entire record. The LZE and LZF keys can be used to delete sections of a record and operate as follows:

– LZE deletes all the characters in the record as of the specified position (including the characters to the right outside of the section displayed on screen). If an entry is made in column 1 of the screen line, a record of length 0 is entered in the work file. LZE can therefore never be used to delete an entire record.

– LZF deletes only the remainder of the screen line; any characters in the record outside of the screen line are pulled in from the right in the next dialog step. If the displayed section comprises the entire record and contains only NULL characters after the delete operation, i.e. it also contains no LZE, then the entire record is deleted and removed from the work file. In this case, LZF has the same effect as the statement code D.

Nondisplayable characters in the text

If a file contains characters which cannot be displayed on screen then these characters are output as the device-specific smudge character which is set as the SUBSTITUTE-CHARACTER in /MODIFY-TERMINAL-OPTIONS.

If such a record is modified, the original character, not the smudge character, is entered in the file. If the position of the smudge character in the record changes due to insertion or deletion (EFG / AFG) then a question mark '?' is entered at the position of the smudge character and the line is displayed in protected mode with a '?' in the statement code column. The original content of the record remains intact.

Warning

If insertion or deletion results in the position of the smudge character moving to a location at which another smudge character was previously present, EDT cannot tell whether the character in question is the original or the displaced smudge character. In this case, the change is not indicated by a question mark '?' and the content of the record may change in ways which were not intended.

Note

In LOWER OFF mode, lowercase letters in the file are output as smudge characters. This is intended to remind users that they have activated the wrong mode. Texts which contain nondisplayable characters should entered in HEX mode (see @PAR HEX) or via the substitute representation (see below).

5.1.1.4 Statement codes in F mode

Statement codes are single-character statements. They are entered in the statement code column. Statement codes are not case-sensitive.

The summary below presents the statement codes by thematic group.

Statement codes used to position the work window

Statement code	Function
+ / -	Position the work window (vertically)
+ / - F1	Position the work window in accordance with the structure depth
S	Position the work window interactively (horizontally and vertically)

Statement codes used to copy and move records

Statement code	Function
*	Delete the copy buffer
C	Collect lines for copying
R	Collect lines for multiple copying
M	Collect lines for moving
A	Copy/move after a line
B	Copy/move before a line
O	Copy/move on a line range (O = on)

Statements codes for record processing

Statement code	Function
D	Delete records
J	Join two records
L	Convert records into lowercase
U	Convert records into uppercase
X	Modify records
H	Display/modify records in hexadecimal mode
E	Insert characters
1..9	Insert data lines
I	Activate the permanent insert function

Statements codes used to handle record marks

Statement code	Function
D F3	Delete a record mark
1..9 F3	Set a record mark

Other statement codes

Statement code	Function
K	Copy a line to the statement line
T	Syntax test by SDF

A detailed description of the individual statement codes can be found in chapter "Statement codes in F mode (alphabetical)" on page 569.

Syntactic and semantic checks

The first step before processing the entries in a work window is to check the syntax and semantics of the statement codes (see the section describing the processing sequence). If invalid statement codes or invalid combinations (e.g. `M` followed by `C`, see below) are found then the subsequent input processing steps are not performed. A `'?'` is output in place of the invalid statement codes and the cursor moves to the first invalid statement code.

Permitted combinations of statement codes in a work window

A distinction is made between the following cases when processing the statement code depending on the function key used for data transfer and/or the entered statement codes.

1. If F3 is used then EDT only accepts statement codes which can be sent using F3 (statements for setting and deleting record marks). These can be combined in any desired way. If illegal statement codes are sent with F3 then EDT considers these to be invalid, marks them with `'?'` and aborts the further processing of the input.
2. If F1 is used then EDT only accepts statement codes which can be sent using F1 (+ or − for positioning on the basis of the structure depth). Only one of these is permitted per work window. If illegal statement codes are sent with F1 then EDT considers these to be invalid, marks them with `'?'` and aborts the further processing of the input.
3. If the statement codes are sent with DUE or a function key other than F1 or F2 then the table below indicates which statement codes may be combined within a work window. When reading the table, it should be borne in mind that a statement code indicated in a row in the table should be entered in the statement code column of the same work window above the statement code indicated in the table column. Statement

codes can be combined (in this sequence) if there is no entry at the intersection between the two codes in the table. An X at the intersection means that they cannot be combined. Special cases are indicated by a lowercase letter and are annotated below.

	+	*	-	A	B	C	D	E	H	I	J	K	L	M	O	R	S	T	U	X	1..9
+	X		X														X	a			
*****		X		b	b										b						
-	X		X					X	X	X							X	a		X	X
A		b																			
B		b																			
C														X		X					
D																					
E	X		X														X	X			
H	X		X														X	X			
I	X		X							X							X	X			
J																					
K												X									
L																					
M						X										X					
O		b																			
R						X								X							
S	X		X					X	X	X							X	X		X	X
T	X		X					X	X	X							X			X	X
U																					
X	X		X														X	X			
1..9	X		X														X	X			

a) If a syntax error occurs in the SDF statement tested with T then + or − ignored.

b) If neither C nor M nor O is specified at the same time as * then the message EDT5360 is issued informing the user that the copy buffer has been emptied and can no longer be copied.

Processing sequence during the processing of statement codes

If multiple statement codes that can be combined with one another are entered in a work window's statement code column then they are processed in the following sequence:

– all D statement codes
– the * statement code for deleting the copy buffer
– the K statement code
– all C, M and R statement codes for making entries in the copy buffer
– all U and L statement codes
– all J statement codes for joining two records
– all A, B, O statement codes for copy and move operations
– all T codes for testing SDF syntax
– the + and − statement codes for positioning
– the S statement code
– all the following statement codes: X (modify), H (modify, hexadecimal), E (insert characters), 1..9 and I (insert lines)

The statement codes X, H, E and I as well as 1..9 are processed from top to bottom in a work window. The statement codes X, H and E after I or 1..9 may be lost if the lines can no longer be displayed on screen due to the insertion area. No warning is issued.

The statement line is evaluated after the statement codes have been processed (see section “Processing sequence” on page 115).

5.1.1.5 Statement in data window – splitting a record

@PAR SEPARATOR can be used to define a freely selectable record separator. If this record separator is entered in a screen line in the data window then the record is split at this point.

Several split points can be defined in one and the same record. The first part of the record retains the originally assigned line number. The following record parts are inserted as new records. Line numbers are assigned using the procedure *Insertion between two lines* (see section “Line number assignment” on page 36).
When inserting the record separator it is necessary to make sure that no characters are lost at the end of the data window line.

Records are only split when new records are inserted or existing records are modified, e.g. when the separator character is inserted or overwritten. It is important to note that copying or moving a record does not cause it to be split, even if the record contains separator characters.

5.1.1.6 Statement line

Entries in the statement line are interpreted as statements. For an overview of the F mode statements, see the section “Statements in F mode” on page 125. The EDT statement symbol (default value: `@`) does not have to be specified (except in the case of the `@:` statement).

Users can enter one or more statements (statement sequence) in the statement line. The individual statements must be separated by a semicolon (`';'`). Processing is aborted if an error occurs. An error message is output together with the unprocessed portion of the statement input including the invalid statement.

When the input has been processed correctly, the statement line is deleted from the screen output. The statement # or `n`# can be used to display the last entered statement or the `nth` last entered statement again so that it can be re-issued either in its original form or modified. In this case, at least one character must be overwritten, modified or added. Alternatively, the @SHIH statement can be used to output the buffer containing the statements most recently executed by EDT in work file 9 (see below).

The content of a statement line or the remainder of a line that is no longer required can be deleted with LZF.

A semicolon is not interpreted as a statement separator in literals.

When @EDIT is used to switch to L mode in a statement string then any residual part of the statement sequence is not processed.

The maximum permitted statement length in F mode is smaller than in L mode due to the limitation imposed by the terminal. For more information, see the following section.

Statement line continuation

If, when the screen is sent, the last character in the statement line is not a `NULL` character, EDT assumes that the user needs a continuation area for input. In this case, a second line is made available provided that the work window is large enough so that at least one further data line can be output. EDT places the content of the statement line in the preceding screen line and the now empty statement line is provided as the continuation line. A maximum of two continuation lines can be provided. The maximum length of input is therefore 189 characters for a terminal with 80 columns or 345 for a terminal with 132 columns.

Treatment of NULL characters in the statement line

`NULL` characters at the end of the statement line are ignored. Before the input is analyzed, `NULL` characters in the statement sequence are converted into blanks.

5.1.1.7 Statement buffer

EDT saves the most recent statements entered in F mode in a buffer. This statement buffer can be output using the @SHIH statement (*Show Input History*). The statement code `K` can be used (following output to a work file) to enter the output line containing the required statement in the statement line.

Alternatively, the # or `n`# statement can be used to retrieve the last or `nth` last statement directly into the statement line.

The statement buffer can accommodate a maximum of 2048 statements (independently of their length). No scrolling statements, statements for changing the operating or work mode or the statements @SHIH and # themselves are entered in the statement buffer. In the same way, statements that are not executed (e.g. because of syntax errors) are not entered in the statement buffer. In contrast, statements that are executed are always entered in the statement buffer, even if they report an error.

Statements that are entered in a statement chain are entered separately in the statement buffer. Statements are always entered in the buffer in the form in which they were originally issued and may therefore sometimes be entered in lowercase irrespective of the @LOWER setting. Leading blanks are removed and empty input is ignored.

5.1.1.8 Status display

From left to right, the status display indicates:

– the line number of the first line in the work window (6 digits) or `0000.00` if the work file is empty
– the column number at which the display of records in the data window begins (5 digits)
– the number of the displayed work file (2 digits in parentheses)

The status display cannot be overwritten. If the status display has the format described here (column number: 5 digits, work file number: 2 digits) this shows that EDT is operating in Unicode mode (in compatibility mode, the column number has 3 digits and the work file number 1 digit).

Example

```
.....................................................................0008.00:00001(21)
```

Line `8.00` is the first line in work window `21`.

5.1.1.9 Processing sequence

The input in a work window is processed in the following sequence:

1. Syntactical and semantic check of the input in the statement code column
2. Evaluation of the data window
3. Execution of the statement codes entered in the statement code column
4. Execution of the statements entered in the statement line

If two work windows are present (see also section “Second work window” on page 119) then each of the above steps is performed first for the upper and then for the lower work window.

irrespective of the number of work windows, as long as insert or delete statement codes (`1..9,I,X,H` or `E`) are present in the statement code column of one of the work windows, only data windows and the statement code are evaluated. First of all, the records are taken over from the data window into the file. The statement code column is then evaluated. The content of the statement line remains unchanged and is not evaluated until none of the above-mentioned statement codes is still specified in any of the work windows. In contrast, if the permanent insert function is active (see statement code `I`) then this does not prevent the evaluation of the statement line.

If errors occur during processing then the following applies:

1. If errors are detected while checking the syntax or semantics of the statement code column, the invalid statement codes are overwritten with question marks (`'?'`) and the further processing of the input is aborted.
2. If errors occur during the evaluation of the data window, for example because it is not possible to split a line with @PAR RENUMBER=OFF, then the statement codes in the affected work window are processed. However, the statement line is not processed and is instead displayed again unchanged. Entries in a second work window, if present, are processed normally.
3. If errors occur during the processing of statement codes, for example because it is not possible to insert lines with @PAR RENUMBER=OFF, then the remaining statement codes in the affected work window are processed. However, the statement line is not processed and is instead displayed again unchanged. Statement codes in a second work window, if present, are also processed. Here again, the statement line is not executed.
4. Errors during the processing of entries in the statement line are not detected until statements in the data window or statement codes have already been processed. Statements in a statement sequence continue to be executed until an invalid statement is found. This applies to each work window independently of the other (see also section “Statement line” on page 113).

Note

If, when the screen is split, @PAR SPLIT=OFF is entered in the upper statement line and a statement is entered in the lower statement line then @PAR SPLIT=OFF is rejected with an error message.

If the two work windows are displaying the same or overlapping sections of the same work file, then statement codes and statements in the two work windows may affect one another, for example if a line is to be simultaneously deleted in one work window and transferred to the copy buffer in the other.
Users are therefore advised not to work in this way. In contrast, non-overlapping sections from the same work file can be processed simultaneously in two different work windows without difficulty.

5.1.2 Modifying the work window

The user can modify the format of the work window by

– activating or deactivating the line number display,
– displaying long records in part or in whole in the data area,
– hiding or showing a column counter (”horizontal ruler”),
– displaying one or more work windows on the screen or
– activating or deactivating the hexadecimal display of the data.

When EDT starts, the format of the data within is set as follows by default:

– Line number display activated,
– No complete display of long records,
– No column counter,
– Non-split (only one) work window, and
– No hexadecimal display.

5.1.2.1 Line number display

The @PAR INDEX statement or the @INDEX statement which is only available in F mode can be used to activate or deactivate the line number display in the work window. By default (@PAR INDEX=ON or @INDEX ON), the format with 72 characters per screen line (or 124 characters per screen line if @VDT F2 has been specified), 6-digit line number display and blank as separator are set.

@PAR INDEX=OFF or @INDEX OFF set 80 (or 132) characters per screen line without line number display. In both formats, the first column of each screen line forms the statement code column. If line numbers are displayed, this overlaps with the first column of the line number display. If line numbers are not displayed, this overlaps with the first column of the data window.

5.1.2.2 Outputting long records

The @PAR EDIT-LONG statement or the @EDIT LONG statement which is only available in F mode can be used to modify the screen output. In the case of records which exceed the number of screen columns, it is possible to specify that

– records are fully displayed in the data window – provided that their length permits this (@PAR EDIT-LONG=ON or @EDIT LONG ON)
– only a section consisting of 72, 80, 124 or 132 characters in a record (depending on the @PAR INDEX setting) is displayed in the data window (@PAR EDIT-LONG=OFF or @EDIT LONG OFF).

EDIT-LONG mode functions without any line number display. A record is written continuously over multiple screen lines.

When new records are input at the end of the work file or in an insert area made available in one of the statement codes I or 1..9, a line which is not terminated either with LZE or with NULL characters is combined with the following line to form a record. In this way, it is possible, depending on the screen format, to enter records with a length of up to 3432 characters directly in EDIT-LONG mode. This only applies to *new* records. If an *existing* record is continued at the end of a work file or is continued into the new line range immediately in front of an insert area then the new records are not combined with the existing record. Existing records can only be extended with the statement code E (see below).

If an input line is shortened by converting a substitute representation (e.g. %U20AC) into the corresponding characters it is not combined with the following line even if the original input line does not contain any terminating LZE or NULL characters.

In EDIT-LONG mode, the hardware tabulator (see @TABS statement) only applies to the first screen line in a record. In the case of existing records, it is not possible to use the hardware tabulator to move to a position in the following screen lines. In the case of new records (see previous section), the tabulator positions in all the lines are identical to those in the first screen line.

If F2 is used to set the entire data window to overwritable, any final record that is completely displayed in the data window remains non-overwritable. If such a record is marked with the statement code X then (if possible) the window is positioned in such a way that the record is fully displayed in the data window. If this is not possible then the record continues to be non-overwritable. Records that are so long that they cannot usually be displayed in full cannot be edited in EDIT-LONG mode. This type of record can only be edited after @PAR EDIT-LONG=OFF. It may then be necessary to scroll horizontally (>, <) in order to move the section to be edited into the displayed area.

The statement code column is the first column on the screen. In the case of multi-line records, the statement code must be entered in the first line. If a record is to be extended with the statement code E then an entire line of NULL characters is output in EDIT-LONG mode. If necessary, the window is repositioned to make this possible. The statement codes S and H are not permitted in EDIT-LONG mode.

Unlike the display with @PAR EDIT-FULL=ON, in EDIT-LONG mode it is only ever possible to overwrite either the statement code column or the associated record in the data window.

In EDIT-LONG mode, neither the column counter set with @PAR SCALE=ON nor an information line requested with @PAR INFORMATION=ON are displayed. The column counter and information lines are not displayed until EDIT-LONG mode is quitted.

If EDIT-LONG mode is quitted with @PAR EDIT-LONG=OFF or @EDIT LONG OFF then the line number display remains active.

EDIT-LONG mode is also deactivated by @PAR INDEX=ON, @PAR INDEX=OFF and @PAR HEX=ON.

Note

The statements for horizontal scrolling (>, <) and the @PAR EDIT-FULL=ON statement are accepted and processed. However, the effects are not visible until EDIT-LONG mode is exited or the line number display is restored.

5.1.2.3 Column counter

The @PAR SCALE=ON statement or the @SCALE ON statement which is only available in F mode can be used to output a column counter (horizontal ruler) in the work window. The column counter is displayed as the 1st screen line in the work window (not in EDIT-LONG mode). The statements @PAR SCALE=OFF or @SCALE OFF deactivate the display of the column counter again.

If a tabulator has been defined (see @TABS), the column counter is extended by an additional screen line in which the current position of the tabulator is displayed.

If the screen is split (see @PAR SPLIT), the @SCALE statement only applies to the work window in which @SCALE was entered.

In EDIT-LONG mode, the @PAR SCALE statement is accepted and processed. However, the effects (showing or hiding the column counter) are not visible until EDIT-LONG mode is deactivated.

5.1.2.4 Second work window

The @PAR SPLIT statement or the @SPLIT statement which is only available in F mode can be used to activate or deactivate the display of a second work window on the screen. The following applies when two work windows are displayed:

Each work window has its own statement line.

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

If a statement is entered in both statement lines, then the cursor is positioned in the upper statement line. If an error occurs while processing a statement then the cursor is positioned in the statement line in which the invalid statement was entered.

If, when the screen is split, @PAR SPLIT=OFF is entered in the upper statement line and a statement is entered in the lower statement line then @PAR SPLIT=OFF is rejected with an error message.

Display in work windows with different character sets

Splitting the work window makes it possible to display two work windows which contain data in different character sets. However, the terminal is only able to display one character set correctly. In addition, changing the character set at the terminal always results in the screen being deleted and then reconstructed (and should therefore be done as infrequently as possible).

EDT therefore attempts to operate without switching the terminal character set if at all possible. If the terminal supports Unicode character sets and automatic character set selection is active, `UTF8` is used for communication with the terminal and this character set is not changed when the window is split (for details, see the introduction to section “The work window” on page 103). In this Unicode character set, the display in the two work windows is correct or, at least, legible.

If a 7-bit or 8-bit character set is used for communication with the terminal, perhaps because the terminal does not support Unicode, then, by default, the upper work window determines the character set used for communication with the terminal unless the user has explicitly modified this setting using the @CODENAME statement.

In a data window containing an assigned work file which is not present in the terminal's character set, the characters are displayed using their equivalents in the terminal's character set or by smudge characters (if the character is invalid in this character set).

The character set used for communications with the terminal can be modified using the @CODENAME statement (format 2).
This statement only modifies the display, not the coding of the data in the work file (see the description of the @CODENAME statement).

For the procedure used to interpret the input, see the introduction to the section “The work window” on page 103 as well as section “Character sets” on page 47.

5.1.2.5 Hexadecimal mode

In F mode, EDT can display the content of work files in hexadecimal form and make it available for editing in this format. This display mode is known as hexadecimal mode or HEX mode for short. It is activated using the statement @PAR HEX=ON or the @HEX ON statement which is only available in F mode. @PAR HEX=OFF or @HEX OFF deactivates HEX mode again. The statement code H can also be used to display an individual line in hexadecimal form and make it available for editing in this format.

The HEX mode display is dependent on the character set in which the data in the work file is coded. In the case of 7-bit and 8-bit character sets, each character is coded by two hexadecimal digits. UTF16 requires four hexadecimal digits for each character while in UTF8 the number of digits required per character varies between two, four and six and in UTFE the number varies between two and eight. In addition, the printable form of the line is always displayed.

The first screen line displays the record content in printable form. Below each printable character in this screen line, the character's hexadecimal code is displayed vertically over multiple screen lines. These screen lines are referred to as hex lines below. There are two hex lines when displaying 7 and 8-bit character sets, four when displaying UTF16, six for UTF8 and eight for UTFE. They are followed by a column counter as in the case of @PAR SCALE=ON. Any column counter which may have been activated for a data window using @PAR SCALE=ON is not output in HEX mode.

Only hexadecimal values (0..9,A..F) and NULL characters are displayed in the hex lines and these, together with blanks, are also the only values permitted for input. Null bytes at the end of a record are not removed and records which are still empty after the end of the file and which are still displayed in the data window are also displayed by means of NULL characters in the hex lines.

HEX mode applies to the current work file independently of the work window, i.e. if the same work file is displayed in different work windows then the same display is used in both work windows.

HEX mode is also deactivated by specifying @PAR EDIT-LONG=ON.

Modifying records in HEX mode

Modifications can be made both in the first screen line (printable form) or in the hex lines. If characters are entered in both the character display and in the hex lines in the same record in a single dialog step then the changes in the character display are ignored.

In the case of 7 or 8-bit character sets, only hexadecimal characters (`0..9, A..F`) may be entered in the hex lines and, in the case of Unicode character sets, the characters must present a legal coding.
If invalid characters or codings are entered then a *correction dialog* opens, i.e. the screen is output again with the invalid character or coding overwritten with a question mark. The cursor is located in the first invalid screen line. The remaining, valid characters or codings are displayed in modified form on the screen but are not yet taken over into the work file. If the user does not want to correct the invalid character then it is possible to exit the *correction dialog* by sending the screen unchanged with DUE. The old content of the invalid line is then restored and the VALID lines are taken over into the work file. In contrast, K3 does not exit the *correction dialog* but simply cancels the last change made by means of keyboard input. Hexadecimal characters may also be deleted (overwritten with `NULL` characters) or removed. When removing such characters, it is important to make sure that the half-bytes in the character codings are not disarranged.

Split screen display

When the screen is split (@PAR SPLIT), the display in `HEX` mode depends on the number of data lines (screen lines in the data window) in the relevant screen:

– If only one screen line is missing then the column counter line is not output.

– If there is not sufficient space to display all the hex lines then only the character display line is displayed. Unlike in the compatibility mode display, the column counter line is not output in this display mode since it is not of use here.

– Any remaining screen lines are used for the display of the following records. These rules then apply recursively to such lines.

Since in `HEX` mode, records are always displayed by means of multiple screen lines, this mode is only of any value if at least one record can be displayed together with all its hex lines. If this is not the case, the message `EDT2404` is output when `HEX` mode is activated. However, `HEX` mode is activated nevertheless. The user can then enlarge the data window to view the hex lines.

Special considerations when displaying UTFE and UTF8

In the subsets of UTF8 and UTFE supported in BS2000, characters are coded using one to three or one to four bytes per character respectively. Despite this, each record is displayed with six or eight hex lines. In the case of characters that are coded with fewer than six or eight bytes respectively, the remaining hex lines contain NULL characters.

Example

The data in the example is coded in UTF8.

```
   1.00 Price change:<·····················································
        57667C66677663·····················································
        025933E4525E7A·····················································
        ·····A·····························································
        ·····4·····························································
        ···································································
        ···································································
        ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   2.00 <·····························································
        ···································································
        ···································································
        ···································································
        ···································································
        ···································································
        ···································································
        ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   3.00       200,00 €<·····················································
        2222223332332E·····················································
        000000200C0002·····················································
        ·············8·····················································
        ·············2·····················································
        ·············A·····················································
        ·············C·····················································
·····························································0001.00:00001(00)
```

5.1.3 Function keys in F mode

In EDT's F mode, it is possible to initiate a large number of actions using function keys. The K2 key is exceptional here since, on the one hand, it is the only function key that also operates in L mode and, on the other, its action can be completely suppressed, for example in noninterruptible procedures or when EDT has been called as a subroutine (e.g. in the POSIX shell).

5.1.3.1 The F keys

All the F keys transfer the input in the data window, the statement code column and statement line from the terminal to EDT. In addition, the keys F1 to F3 have special functions:

F1 Positioning at records with the same structure depth

F1 can be used in combination with the statement codes + and − to move to the next record with the same structure depth (see section “Statement codes in F mode” on page 109).

F2 Setting all the screen lines in the data window to overwritable

If the screen is sent with F2 then the data window, or both data windows if the screen is split, is set to overwritable on the following output.

When transfer is performed with F2, if the entries in the statement line have not yet been executed due to the processing sequence observed by EDT, for example because both a statement and one of the statement codes 1..9, I or E (see the section dealing with the processing sequence) have been transferred with F2 then the data window is first set to overwritable and the statement line is output again unchanged. The changes then entered in the data window become effective even before the processing of the statement line.

F3 Processing record marks

F3 in combination with the statement codes 1..9 and D or the scrolling statements +, − and ++, −− triggers the following functions:

– Set record marks (statement codes 1..9)
– Delete record marks (statement code D)
– Move to records with record marks (statements +,−, ++, −−)

5.1.3.2 The K keys

Unlike the F keys, pressing a K key does not trigger any transfer of the modified data from the screen to EDT. All input in the screen is therefore lost.

K1 Terminating EDT

Pressing K1 requests the termination of EDT. Unlike termination with @HALT, a confirmation query is issued in the work window's message line even if the work files do not contain any unsaved data:

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

If the user enters `Y` then EDT is terminated. If `N` is entered then EDT continues to run. If one of the work files contains unsaved data then K1 has the same effect as @HALT (see also section “Terminating an EDT session” on page 92).

K2 Interrupting the EDT session

It is possible to interrupt the EDT session and switch to system mode by means of the @SYSTEM statement or by pressing K2.

The `/RESUME-PROGRAM` command can be used to return to F mode. After this, the entire screen is output again.

If the work window in which the EDT session was interrupted is not output or is output incompletely after `/RESUME-PROGRAM` then the original content can be restored with K3.

If, during the period of interruption, another program is loaded (e.g. with `/START-PROGRAM` or `/LOAD-PROGRAM`) or if a procedure is started which loads another program then EDT is unloaded without any query being issued.

K3 Restoring the screen content, rejecting user input

If the screen content is moved (for example, due to a broadcast message), then K3 can be used to restore the original state. The screen content (together with any messages output by EDT) is restored exactly as it was before the user entered the first character. The K3 key can therefore also be used to reject all user inputs and restart entry, for example if the user has accidentally overwritten screen lines which should have remained unchanged with new text.

K4 to K15 are treated in the same way as K3 (see above).

5.1.4 Statements in F mode

The following statements are permitted in F mode

<	@GETJV	@SEARCH-OPTION
<<	@GETLIST	@SEPARATE
+	@GETVAR	@SEQUENCE
++	@HALT	@SET
$0..$22	@HEX	@SETF
-	@INDEX	@SETJV
--	@INPUT (Format 1+2)	@SETLIST
>	@LIMITS	@SETSW
@:	@LIST	@SETVAR
#	@LOAD	@SHIH
@AUTOSAVE	@LOG	@SHOW
@BLOCK	@LOWER	@SORT
@CHECK (Format 2)	@MODE	@SPLIT
@CLOSE	@MOVE	@STAJV
@CODENAME	@ON	@STATUS
@COLUMN	@OPEN	@SUFFIX
@COMPARE	@P-KEYS	@SYMBOLS
@CONVERT	@PAGE	@SYNTAX
@COPY	@PAR	@SYSTEM
@CREATE (Format 1+ 2)	@PREFIX	@TABS
@DELETE	@PRINT	@TMODE
@DELIMIT	@QUOTE	@UNLOAD
@DO (Format 1)	@RANGE	@UNSAVE
@DROP	@READ	@USE
@EDIT	@RENUMBER	@VDT
@ELIM	@RESET	@VTCSET
@END	@RETURN	@WRITE
@ERAJV	@RUN	@XCOPY
@EXEC	@SAVE	@XOPEN
@FSTAT	@SCALE	@XWRITE
@GET	@SDFTEST	0...22

In F mode, the EDT statement symbol (default value: @) does not have to be specified (except in the case of the @: statement). For a detailed description of the statements, see chapter 9.

5.2 L mode

In L mode, files are processed line-by-line, that is to say that in interactive mode, EDT only outputs one line (the *current* line) at a time or only reads one line (in both batch and interactive mode) from SYSDTA. This line may contain both records and statements and is processed as soon as it has been read in.

Records are written to the current line and the current line is then increased by the current increment. The current line can be addressed symbolically via the '*' symbol, e.g. @PRINT *.

Statements are executed immediately. The EDT statement symbol @ is used for differentiation (see below).

L mode is available in both interactive and batch mode.

When @DO procedures and @INPUT procedures are run as well as when SYSDTA is read using RDATA (system procedures, batch mode), the statements are processed as if they had been entered in L mode. In such cases, only L mode statements are therefore permitted.

The @EDIT FULL statement is used to switch to F mode.

5.2.1 Input in L mode

EDT interprets input in L mode as a statement if

– the first character other than a blank is the EDT statement symbol (default value: @) or a user statement symbol for an external statement routine (see @USE statement)
– and the second character other than a blank is not identical to the first character that is not a blank.

If EDT recognizes a statement then it is executed immediately.

In the next two paragraphs, the term *statement symbol* is used to refer to both the *EDT statement symbol* and the *user statement symbol*.

All input in which the first character other than a blank is not a statement symbol is interpreted as a record and is stored unchanged in the current line. In interactive mode, empty input which is sent with F1 instead of DUE causes an empty line (line of length 0) to be stored in the current line (see section “Function keys in L mode” on page 128).

Input in which the first two characters other than blanks are statement symbols are also interpreted as records but are subject to special processing:
The characters located before the second statement symbol (which can only be blanks or the first statement symbol), are removed.
This special processing simplifies the creation of procedures in L mode (see also section "EDT procedures" on page 64). The input is not immediately executed as a statement but is stored as a statement and can therefore be subsequently run as many times as necessary.

Examples

Input	**Interpretation**
`@RENUMBER`	Statement
`␣␣␣␣␣␣@␣␣␣␣␣␣␣RENUMBER`	Statement
`RENUMBER`	Record, the value 'RENUMBER' is stored
`@@RENUMBER`	Record, the value '@RENUMBER' is stored
`@␣␣␣␣␣␣␣␣@RENUMBER`	Record, the value '@RENUMBER' is stored
`␣␣␣@␣␣␣␣␣@␣␣RENUMBER`	Record, the value '@␣␣RENUMBER' is stored

5.2.2 Entering records in character, hexadecimal or binary format

In L mode, records (but not statements) can be entered not only as character strings but also as sequences of hexadecimal or binary characters. The @INPUT statement (format 3) is used to switch between these input formats.

By default, EDT expects L mode input in the form of character strings (@INPUT CHAR). In this format, the character set of the underlying data source is used (terminal, SYSDTA, file, library element, work file). Since the entered records are inserted in the current work file and the character set used in this work file may be different from that of the data source, a conversion operation may be necessary. The precise rules are described in section "Character sets" on page 47.

If records are to be entered as sequences of hexadecimal or binary characters in L mode, then this must be set explicitly using the statement @INPUT HEX or @INPUT BINARY. The hexadecimal or binary characters themselves are expected in the character set used in the associated input source. The specified codes are then interpreted in the character set used in the current work file. If hexadecimal characters are entered which do not correspond to any valid character in this character set then the input is rejected with the message EDT5460 (see also section "Character sets" on page 47).

Once hexadecimal or binary input has been activated, only records in a valid hexadecimal or binary format are accepted. Invalid input is rejected with the message EDT3902 or EDT3901. If the number of entered characters is not a multiple of 2 or 8 then the input is left-filled with blanks.

Since, when a hexadecimal representation is used, each character is coded by at least two bytes, the number of characters that can be entered per line is reduced to no more than half of the value that would otherwise apply to the input source in question. In the case of binary representation, the number of characters is reduced to an eighth or less of the value applicable for the input source.

Example

```
1.     ABC
2.     @INPUT HEX
2.     C1C2C3
3.     @INPUT BINARY
3.     110000011100001011000011
4.     @PRINT
1.0000 ABC
2.0000 ABC
3.0000 ABC
4.
```

5.2.3 Function keys in L mode

The following description relates to interactive mode only.

In L mode, all the F keys have the same effect as DUE, irrespective of whether RDATA (input prompt is *) or WRTRD (input prompt is the line number) is used for reading.

The DUE2 key transfers the entire screen content including any LZE characters and end marks that may be present. In EDT, this system characteristic may cause multiple lines to be inserted in the current work file. Since this is not usually desired, the DUE2 key should not be used for entry in L mode.

In L mode, the F1 key is used to enter a blank line (line of length 0). To do this, F1 or EM F1 should be entered at the input prompt. In contrast, if only DUE or EM DUE is entered, the input is ignored (as in the past) and the input prompt is displayed again. If other characters are entered in addition to F1 or EM F1 then F1 has the same effect as DUE.

If any of the K keys other than K2 is entered then all the characters entered at the screen are ignored and the input prompt is displayed again.

K2 causes EDT to be interrupted and processing switches to system mode unless this is prevented by the system settings (see section “Interrupting an EDT session” on page 91).

5.2.4 Statements in L mode

The following statements are permitted in L mode

@+	@GETVAR	@SEARCH-OPTION
@–	@HALT	@SEPARATE
@:	@IF	@SEQUENCE
@AUTOSAVE	@INPUT	@SET
@BLOCK	@LIMITS	@SETF
@CHECK	@LIST	@SETJV
@CLOSE	@LOAD	@SETLIST
@CODENAME	@LOG	@SETSW
@COLUMN	@LOWER	@SETVAR
@COMPARE	@MODE	@SHIH
@CONTINUE	@MOVE	@SHOW
@CONVERT	@NOTE	@SORT
@COPY	@ON	@STAJV
@CREATE	@OPEN	@STATUS
@DELETE	@P-KEYS	@SUFFIX
@DELIMIT	@PAGE	@SYMBOLS
@DIALOG	@PAR	@SYNTAX
@DO (Format 1)	@PREFIX	@SYSTEM
@DROP	@PRINT	@TABS
@EDIT	@PROC	@TMODE
@ELIM	@QUOTE	@UNLOAD
@END	@RANGE	@UNSAVE
@ERAJV	@READ	@USE
@EXEC	@RENUMBER	@VDT
@FILE	@RESET	@VTCSET
@FSTAT	@RETURN	@WRITE
@GET	@RUN	@XCOPY
@GETJV	@SAVE	@XOPEN
@GETLIST	@SDFTEST	@XWRITE

The following statements are *not* permitted in EDT procedures:
@DROP, @DIALOG, @INPUT (format 1 and 2)

The following statements are *only* permitted in EDT procedures:
@GOTO, @DO (format 2), @PARAMS

The EDT statement symbol (default value: @) must be specified in L mode.
For a detailed description of the statements, see chapter 9.

6 File processing

EDT can be used to process ISAM files, SAM files, library elements and POSIX files. These files are loaded into work files for processing. When the term “file” is used below, it is intended to refer to all four of these file types.

EDT also provides input/output interfaces to the BS2000 system files SYSDTA, SYSOUT and SYSLST.

The handling of character sets on read and write operations is described in section “Character sets” on page 47. Files with character sets which are not known in XHCS cannot be processed.

6.1 File types

The file types supported by EDT are described in the following sections.

6.1.1 SAM files

When accessing an existing SAM file, any printer control character specification in the catalog is ignored. Moreover, in the case of SAM files with variable record length, the explicit specification of a record length in the catalog is ignored.

When EDT writes to a new SAM file, this file is usually stored on disk with the following default attributes:

– variable record length without record length specification,
– block size 2 for files on NK4 disks or block size 1 otherwise. If when the file is opened, the longest record that is to be written to it does not fit in this block size then a larger block size (maximum 16) is used.

If the attributes of new files shall differ from these default values then the file attributes and a file link name must be stored in the Task File Table before the file is written and the write operation must be performed using this file link name.
However, only the attributes described in section “File link names” on page 139 are evaluated.

Usually, the files that are to be processed by EDT are read (fully or partially) into work files. It is not possible to process SAM files directly on disk (real processing). However, EDT is able to copy SAM files to ISAM files and then open these for real processing (see the @OPEN statement, format 2, page 411).

File names used in EDT statements must comply with the requirements of the BS2000 data management system. EDT checks the validity of file names.

EDT is able to process records of length `0`.

SAM files on magnetic tape cannot be processed using the @OPEN (format 1) and @CLOSE statements. In this case, the other statements (@COPY, @WRITE) should be used. If a new SAM file is to be created on magnetic tape, its name and properties must first be declared using the BS2000 `/CREATE-FILE` command.

It is not possible to process SAM files with record format `UNDEFINED` in EDT.
Similarly, attempts to process SAM files with the character set `UTF16` and a fixed, odd-numbered record length are rejected.

6.1.2 ISAM files

When accessing an existing ISAM file, any printer control character specification in the catalog is ignored. Moreover, in the case of ISAM files with variable record length, the explicit specification of a record length in the catalog is ignored.

When EDT writes to a new ISAM file, this file is usually stored with the following default attributes:

– variable record length without record length specification,
– key position 5,
– key length 16 in the case of files with the character set `UTF16` or key length 8 in the case of files with a different character set and
– block size 2 for files on NK4 disks or block size 1 otherwise. If when the file is opened, the longest record that is to be written to it does not fit in this block size then a larger block size (maximum 16) is used.

By default, no multiple keys are permitted on writing.
If the attributes of new files shall differ from these default values then the file attributes and a file link name must be stored in the `Task File Table` before the file is written and the write operation must be performed using this file link name.
However, only the attributes described in section “File link names” on page 139 are evaluated.

Usually, the files that are to be processed by EDT are read (fully or partially) into work files. It is also possible to process ISAM files directly on disk (real processing). This can only be done in work file 0.

File names used in EDT statements must comply with the requirements of the BS2000 data management system. EDT checks the validity of file names.

EDT is able to process records which consist solely of the ISAM key.

When ISAM files are processed, the ISAM key can be read in as a line number, read into the work file's data area (as line content) or completely ignored.

On a write operation, the ISAM key can be formed from the line number or taken over from the data area.
If the ISAM key is to be retained it must therefore first be taken over as a line number or into the data area and must not be modified by EDT statements.

If the ISAM key is taken over as a line number when read in, it must be numerical (each of the 8 characters belongs to the range `0..9`) and the key `00000000` is not permitted.

If the file's key length is shorter than 8 (or 16 in the case of `UTF16` files) and if the line number is formed from the ISAM key then the line number is left-filled with zeros. For example, in the case of a `KEY-LENGTH` specification of 4, the ISAM key `1234` would be taken over as line number `0000.1234`.

If the file's key length is shorter than 8 (or 16 in the case of `UTF16` files) and if the ISAM key is formed from the line number then the line number is truncated at the left. For example, in the case of a `KEY-LENGTH` specification of 4, the line number `1234.5678` would be written as the ISAM key `5678`. It is not therefore always possible to guarantee that the ISAM key is unique. If multiple keys are permitted when writing to the file, then work file records for which the same ISAM key was generated are written to the file. Otherwise, the write operation is aborted and the message `EDT4208` (DMS error code `0AAF`) is output.

The only way to process ISAM files with non-standard key positions (≠5 for variable record lengths or ≠1 for fixed record lengths), with a key length greater than that specified as the default, with non-numerical keys or with duplicate key values is to take the ISAM key over into the data area.

If the ISAM key is located in the data area and modified in EDT, if lines are swapped or records inserted, then the user must make sure that the sequence of work file records corresponds to the sequence of ISAM keys as otherwise the write operation will be rejected with the message `EDT4208` (DMS error code `0AAB`).

In the case of ISAM files with duplicate key values (`duplicate keys`), it is possible to read in all the records provided that the key is not taken over as a line number. If the ISAM key is taken over as a line number then only the last record with the same key is read in.
In order to process such files, the key should therefore also be taken over into the record.
When files with multiple keys are written, the attribute `DUPLICATE-KEY=YES` must be stored in the `Task File Table`.
It is not possible to process ISAM files with the character set `UTF16` and a fixed, odd-numbered record length or an odd-numbered key length.

The definitions of any secondary keys in an ISAM file (in a secondary index) are retained if the file is processed using the @OPEN and @CLOSE statements and the key fields are not modified inconsistently in the data area.

If inconsistent changes are made, the message EDT5246 is output and the secondary index is deleted. The secondary index is also deleted if the file is fully overwritten with the @SAVE or @WRITE (format 2) statement (but not in the case of UPDATE).

6.1.3 POSIX files

The POSIX subsystem must be activated before it is possible to process files in the POSIX file system.

The POSIX file names in the statements which permit access to POSIX files are path names in the POSIX file system. EDT cannot be used to move to a position within the POSIX file system. If no absolute path name (starting with '/') is specified then the file names always refer to the current directory. When the call is issued from BS2000, this is the user's home directory.

POSIX file names used in EDT statements must comply with the requirements of the POSIX file system. In particular, the maximum permitted length is 1023 bytes. It can be specified directly as a string or indirectly as a string variable. If the name of a POSIX file contains blanks or other special characters then it must be specified by means of a string variable.

POSIX file names are case-sensitive. Consequently, when input is made from a terminal, @PAR LOWER=ON should be activated.

EDT reads data character-by-character. The end of a record is recognized by means of the (character set-specific) end-of-record character (for example X'15' in EBCDIC or X'0A' in ISO character sets). If a record is detected, it is placed in the current work file. If two end-of-record characters occur one after the other then a record of length 0 is generated in the work file.

If when reading a POSIX file, no end-of-record is recognized after 32768 characters then EDT outputs the message EDT1253, truncates the output and ignores the characters through to the next end-of-record character.

When the content of the work file is written to the POSIX file, a (character set-specific) end-of-record character (for example X'15' in EBCDIC or X'0A' in ISO character sets) is written after every work file record.

In the case of records of length 0, only an end-of-record character is written.

When writing write-protected POSIX files under the user ID TSOS in interactive mode, the message EDT0244 is issued to ask the user whether write access is to be permitted. If it is not, EDT issues the message EDT5312. In batch mode, EDT issues the message EDT5312 and does not permit write access.

Unlike in the case of BS2000 files or library elements, the system permits the multiple opening of the same POSIX file. EDT does not prevent this. Users must therefore take the necessary care and attention.

6.1.4 Library elements

EDT is able to process elements in program libraries (PLAM libraries, *Program Library Access Method*). For more detailed information on these libraries, see the LMS User Guide [14].

In the current manual, program libraries are simply referred to as libraries.

Delta elements cannot be processed using EDT statements. Although read access is possible, if users want to modify delta elements then they must use the LMS statements in EDT. The precise procedure is described in the LMS User Guide [14].

The library name in EDT statements must comply with the BS2000 data management system requirements for file names. It can be specified directly as a string or indirectly as a string variable. If the library that is to be opened is a part of a file generation group then the library name must be specified by means of a string variable (because of the parentheses).

EDT is able to process records of length `0`.

Elements in libraries can be addressed individually via their element designations.

The element designation consists of the name, version and element type and is specified in the following form:

```
elname[(vers)][,eltype]
```

Here, `elname` is the name of the element, `vers` the version designation of the element and `eltype` the type of element.

The version and element type specifications are optional. If no value is entered for `vers` in a statement then the highest existing version is used on read operations and highest possible version on write operations. If no value is entered for `eltype` in a statement or if the value is `*STD` then the value specified in @PAR ELEMENT-TYPE is used. The presetting when EDT is started is `S`.

The element designation must comply with the name conventions as defined in the LMS User Guide [14].

In the read and write operations, it is only possible to specify text-type and user-defined types as the element type. The text-type standard types are:

Type	Element content
S	Source program
M	Macros
P	Data edited for printing
J	Procedures
D	Text data
X	Data in any format

In the case of the standard types, no check is performed to determine whether the content of the element is actually text-type. If the element type is a user-defined type, no check is performed to determine whether it is derived from a text-type basic type.

Elements containing format B records cannot be edited by EDT. In the case of format A records, EDT only takes account of record type 1 records. EDT leaves other record types unmodified.

6.2 Basic information on reading and writing data

If a file that is to be read contains characters which are not supported by the work file's character set then the file is not read in unless a substitute character has been defined which is then used. If the file contains an invalid byte sequence (possible in Unicode character sets) then the file is not read in.

Since EDT provides internal support for lines of up to 32768 characters, it is possible, when writing to DMS file or library elements, that a record may not be written because it is too long. In particular, when writing to a file which uses the `UTF16`, `UTFE` or `UTF8` character set, this may occur in connection with significantly shorter lines if the work file record contains sufficient characters that are coded as 2-byte or 3-byte characters. This phenomenon can therefore easily occur in files with a fixed record length. If it is not possible to write a record because it is too long, EDT outputs the message `EDT5444` and aborts the write operation. It is then the user's responsibility to shorten the lines accordingly or use another file format.

If when writing a file with fixed record length, a work file record is shorter than the file's record length, EDT fills the record with blanks up to the defined record length.

In the case of DMS files, the required file size is estimated before writing and, if possible, the required number of pages is assigned as the *primary allocation* in order to prevent the file from being split into a large number of different extents.

Once a DMS file has been closed after writing (statements @WRITE, @CLOSE, @SAVE), EDT usually releases the unneeded disk storage space. This space is not released if job switch 7 was set before writing or if the file name starts with a user ID.

When write operations are performed using the old statements (see chapter 9) – @WRITE (format 2) and @SAVE – this can also be achieved by assigning the file name a fixed predefined file link name and specifying the symbolic designation `'/'` instead of the file name.

6.3 Reading and writing all supported file types

The following statements can be used to process files belonging to all the file types supported by EDT. It is recommended that you now use only these statements for file processing.

6.3.1 Reading

The @OPEN statement (format 1) can be used to read a file into the current work file for processing or to create a new file. The file remains open until it is closed with @CLOSE.

The @COPY statement (format 1) can also be used to read a file into the current work file. After being read in, the file is closed again.

When ISAM files are read, the KEY operand in the @OPEN or @COPY statement can be used to determine whether the ISAM key is to be ignored or to be taken over as a line number or into the data area.

A file can be run as a procedure by means of the @INPUT statement (format 1).

6.3.2 Writing

The current work file can be written back to a file with @CLOSE provided that this has been previously read in or created with @OPEN (format 1). After the write operation, the file is closed and the work file is deleted.

The current work file can be written to a new file with @WRITE (format 1). The work file is retained after writing.

The current work file can be written to an existing file with @WRITE (format 1). This completely replaces the old file content. The work file is retained after writing.

The current work file can be written back to the file opened with @OPEN (format 1) with @WRITE (format 1). The file remains open after writing and the work file is retained.

If a new SAM or ISAM file is created before the write operation then the file type's default attributes are generally used. If file attributes other than the defaults are to be used then they must be stored in the `Task File Table` together with a freely definable file link name before the @OPEN or @WRITE statement is called and the call to the @OPEN or @WRITE statement must use this file link name.

6.3.3 File link names

When using the new statements, it is possible to employ freely definable file link names to address SAM and ISAM files. Here, it is essential that at least the file name is entered in the `Task File Table`.

In the case of **existing** files only a few of the attributes are taken over from the `Task File Table`. These are (if specified):

```
FILE-NAME
WRITE-CHECK
```

In the case of **ISAM** files, the following attributes are also taken over:

```
WRITE-IMMEDIATE
DUPLICATE-KEY
PADDING-FACTOR
SHARED-UPDATE
```

All the other specifications are ignored and are taken over from the existing file.
In the case of **newly created** files, more of the attributes specified in the `Task File Table` are taken over. These are (if specified):

```
FILE-NAME
ACCESS-METHOD
RECORD-FORMAT
RECORD-SIZE
PRINT-CONTROL
BLOCK-SIZE
BLOCK-CONTROL
WRITE-CHECK
```

In the case of **ISAM** files, the following attributes are also taken over:

```
KEY-LENGTH
KEY-POSITION
WRITE-IMMEDIATE
DUPLICATE-KEY
PADDING-FACTOR
VALUE-FLAG-LENGTH
PROPAGATE-VALUE-FLAG
LOGICAL-FLAG-LENGTH
SHARED-UPDATE
```

All other specifications are always ignored.

The access method specification is only used if there is no explicit specification in the corresponding statement.

6.4 Characteristics of the old file access statements

Preliminary comment

The term **old** does **not** refer to the distinction between these statements in Unicode and compatibility mode but to the file access statements originally developed for EDT. These remain valid in both operating modes.

The old file access statements @READ, @GET, @WRITE (format 2), @SAVE, @ELIM and @OPEN (format 2) are subject to certain restrictions. They are not able to process all file types and certain accesses are only possible using workarounds (definition of file link names). They provide a number of special functions for the supported file types. These are described in the present section.

If a file is read into the current work file using the @READ or @GET statements then the file is closed again immediately after being read in. The file is then not opened again for writing until a @WRITE or @SAVE statement is issued. If the `UPDATE` operand is not specified in these statements, the old file content is deleted and completely replaced. The file is not locked for other users while it is being processed. During a write operation, it is therefore possible that intermediate changes made by other users may be overwritten.

6.4.1 Predefining file names

Following the read operation with @READ or @GET, the work file is linked to the file name (implicit local @FILE entry). This predefines the file name for the @WRITE (format 2) and @SAVE statements.

Linkage can also be performed explicitly using the @FILE statement. The predefined file name then applies to the statements @READ, @GET and @WRITE (format 2), @SAVE, @ELIM and @OPEN (format 2).

The linkage is eliminated again when the work file is completely deleted with @DELETE (format 2) or with statements which implicitly execute a @DELETE (format 2) or is explicitly deleted with the @FILE statement.

In addition, the @FILE statement can be used to define a presetting which applies globally to all the work files. An explicit @FILE entry (including global entries) always has priority over an implicit @FILE entry.

6.4.2 Partial reading and writing

The @READ and @GET statements can be used to select the records which are to be read from the file into the work file. When this is done, it is possible to modify the sequence of records in the file. Records can be read into the work file more than once.

It is also possible to select the characters (columns) which are to be read from the selected records into the work file. Here again, it is possible to modify the character sequence and characters can be read in more than once.
If column values which exceed the record length are specified then blanks are read into the work file in their place. This applies corresponding to the @INPUT statement (format 2).

If line ranges are specified for selection then only the specified (and possibly adjacent) lines are examined for illegal byte sequences. However, they are examined in full independently of any column selection which may have been made. Illegal byte sequences which occur in non-read lines are not detected.

The @WRITE (format 2) and @SAVE statements can be used to select the work file records to be written to the file. When this is done, it is possible to modify the sequence in which the work file records are written.

Work file records can be written more than once. It is also possible to select the characters (columns) which are to be written from the selected work file records into the file. Here again, it is possible to modify the character sequence and characters can be written more than once. If column values which exceed the work file record length are specified then blanks are written to the file in their place.

6.4.3 Version numbers

In the statements @GET, @SAVE, @READ, @WRITE (format 2), @OPEN (format 2), @ELIM, @INPUT (Format 2), @FILE and @UNSAVE it is possible, in addition to the file name, to specify a version number between 0 and 255 or specify `*` to represent the current version number. This makes it possible to protect the file against accidental overwriting.

When a file is first created, it is assigned the version number 1 after being written to disk. Each time the file is written, DMS increases the version number by 1.
The version number is incremented up to 255. The following version number is then 0 again.

Read accesses (@GET, @READ) with a version number other than the current version number are executed. The current version number is output in message `EDT0902`.
If `*` is specified as the version number then the current version number is also displayed in the message `EDT0902`. Although this specification is possible in @INPUT (format 2), it is completely ignored. Write accesses (@SAVE, @WRITE (format 2), @ELIM, @UNSAVE)

and attempts to open files for real processing (@OPEN, format 2) with a version number other than the current value are not executed. Instead, the correct version number is displayed in the message EDT4985.
In the case of the explicit or symbolic specification of the current version number, the new version number which has been incremented by 1 is displayed in the message EDT0902.

The version numbers provide increased protection against file destruction. If a version number is specified when a file is read in, then the valid version number is output after the read operation, thus making it possible to identify obsolete file versions.

The EDT version number should not be confused with the generation number of file generation groups in BS2000. This is a component of the file name.
For each generation, it is also possible to specify a version number.
The meaning of the EDT version number corresponds more closely to the variant number of library elements.

6.4.4 File link names

In the statements @ELIM, @GET, @SAVE, @READ and @WRITE (format 2), it is possible to specify '/' instead of the file name if the fixed file link name EDTISAM (for @ELIM, @GET, @SAVE) or EDTSAM (for @READ, @WRITE) has been assigned to the file before it is accessed by EDT.

If the fixed file link name EDTISAM has been assigned to a file then the attributes are read from the Task File Table when the file is opened for reading or writing with @GET or @SAVE. If EDTSAM has been assigned to the file then the attributes from the Task File Table are used when the file is opened for reading or writing with @READ or @WRITE.

If on a write operation, the file attributes differ from the EDT standard format (see the sections on ISAM files and SAM files) then it is only possible to write the files with the statements @SAVE and @WRITE (format 2) if the discrepant attributes have first been entered in the Task File Table and the fixed file link name EDTISAM or EDTSAM has been assigned to the file.

If the symbolic name '/' is specified for a file when a write operation is performed with @SAVE or @WRITE then the unneeded disk storage space is not released after the file is closed and no confirmation query is issued (EDT0903) asking whether an existing file should be overwritten.

6.5 Reading and writing SAM files with the old statements

The old statements @READ and @WRITE (format 2) can still be used to access SAM files. However, users are recommended only to use the new statements (see section “Reading and writing all supported file types” on page 138).

These statements make it possible to select specific lines and columns and specify a version number. If the file has been assigned the fixed file link name `EDTSAM` then it is also possible to specify `'/'` instead of the file name (see section “Characteristics of the old file access statements” on page 140).

These statements also make it possible to interpret the first 8 characters in a SAM file (in `UTF16` files, these are the first 16 bytes) as a line number.
In the case of write operations, the associated EDT line numbers are used and the work file record is written as of character 9.
On read operations, the first 8 characters of the record that is to be read are taken over as a line number and the remainder of the record as of character 9 is taken over into the work file.
In this case, a check is performed to verify that the characters in the line number are numerical (for details, see the descriptions of the individual statements).

6.5.1 Reading

A SAM file can be read into the current work file for processing using the old @READ statement. After being read in, the file is closed again.

An implicit local @FILE entry is created and this can be used in a subsequent @WRITE statement.

6.5.2 Writing

The current work file can be written to a SAM file using the old @WRITE statement (format 2). This may either exist or can be created before the write operation. The work file is retained after writing.

Existing SAM files with a fixed record length can only be overwritten in the same record format if this file attribute has been stored with the file name and the fixed file link name `EDTSAM` in the `Task File Table` before the @WRITE statement is executed.

If a new SAM file is created before the write operation then the default attributes are generally used. If file attributes other than the default attributes are used then they must be stored together with the file name and the fixed file link name `EDTSAM` in the `Task File Table` before the @WRITE statement.

6.6 Reading and writing ISAM files with the old statements

The old statements @GET, @SAVE and @ELIM can still be used to access ISAM files. However, users are recommended only to use the new statements (see section “Reading and writing all supported file types” on page 138).

These statements make it possible to select specific lines and columns and specify a version number.
If the file has been assigned the fixed file link name `EDTISAM` then it is also possible to specify `'/'` instead of the file name in the @GET and @SAVE statements (see section “Characteristics of the old file access statements” on page 140).

6.6.1 Reading

An ISAM file can be read into the current work file for processing using the old @GET statement. After being read in, the file is closed again.

An implicit local @FILE entry is created and this can be used in a subsequent @SAVE statement.

The ISAM key is not usually transferred as a line number. However, this is possible by means of the `NORESEQ` operand.
It is not possible to transfer the ISAM key into the data area with @GET. To do this, it is necessary to store the file name, the fixed file link name `EDTSAM` and the ISAM access method in the `Task File Table` and use the statement @READ `'/'`. In this case, it is not possible to specify the file name in the @READ statement.

To read in ISAM files with non-standard key positions (≠5 for variable record lengths or ≠1 for fixed record lengths), with a key length greater than that specified as the default, with non-numerical keys or with duplicate key values, it is necessary to take the ISAM key over into the data area.
To do this, it is necessary to store the divergent file attributes together with the file name, the fixed file link name `EDTSAM` and the ISAM access method in the `Task File Table`. The file is then read in using the @READ `'/'` statement.

6.6.2 Writing

The current work file can be written to a ISAM file using the old @SAVE statement. This may either exist or can be created before the write operation. The work file is retained after writing.

Existing ISAM files with a fixed record length and/or a key length shorter than that defined as the default can only be overwritten using these same attributes if the divergent file attributes have been stored together with the file name and the fixed file link name `EDTISAM` in the `Task File Table` before the @SAVE statement is executed.

Existing ISAM files whose ISAM key has been taken over into the data area can be overwritten if the divergent file attributes have been stored in the `Task File Table` together with the file name, the fixed file link name `EDTSAM` and the ISAM access method and are written using the @WRITE `'/'` statement. In this case, it is not possible to specify the file name in the @WRITE statement.

If a new ISAM file is created before the write operation then the default attributes are generally used.

If the ISAM key is to be formed from the line number when a new file is created then it may be necessary to store the file attributes which differ from the default values in the `Task File Table` together with the file name and the fixed file link name `EDTISAM` before the @SAVE statement is executed.

If the ISAM key is to be taken from the data area when a new file is created then it may be necessary to store the file attributes which differ from the default values in the `Task File Table` together with the file name and the fixed file link name `EDTSAM` and the ISAM access method. The file is then written in using the @WRITE `'/'` statement. In this case, it is not possible to specify the file name in the @WRITE statement.

The @ELIM statement can be used to delete the ISAM file either fully or partially.

6.7 Real processing of ISAM files

ISAM files can be processed directly on disk without first having to be read fully into the EDT memory area. When this is done, work file 0 must be the current work file.
It must either be empty or a file must have been opened for real processing.
In the former case, the character set of work file 0 must be `*NONE` or must correspond to the character set of the file that is to be opened.
In the latter case, the opened file is implicitly deleted. In this case, the work file is implicitly deleted and the character set `*NONE` is set.

6.7.1 Opening

An ISAM file is opened for real processing using the @OPEN statement (format 2). If the file does not yet exist, it is created before being opened. When this is done, the default attributes for ISAM files are used unless a different specification has been made using the fixed file link name `EDTMAIN`. The file remains open until processing is terminated by means of an explicit or implicit close.

If the file link name `EDTMAIN` is used, it is still necessary to specify the file name in the @OPEN statement. It is not possible to specify `''/''`.

The real processing of ISAM files with non-standard key positions or key lengths longer than the default value is not supported.

The real processing of ISAM files with duplicate key values or non-numerical keys is not supported. If such a file is opened then an error occurs as soon as one of its records is processed.

If a file's key length is shorter than 8 (or 16 in the case of `UTF16` files) then the line number is left-filled with zeros when read in. For example, in the case of a `KEY-LENGTH` specification of 4, the ISAM key `1234` would be taken over as line number `0000.1234`. Furthermore, in this case the current increment would be set to a *suitable* value. If when the file is opened, the current increment has the default value 1 or if the set increment is greater than the largest possible line number then it is set to the value `0.0001` for key lengths smaller than or equal to 2, to `0.01` for key lengths smaller than or equal to 5 and to the value `1.0` otherwise.

6.7.2 Processing

Only the section of the ISAM file that is actually required is read into the current work file. In F mode, these are the records which are fully or partially displayed on the screen and, in L mode, the line range specified in an EDT statement. On a read operation, the ISAM key is taken over as a line number.

In F mode, every change is immediately transferred to the disk file when the DUE key or another data transfer key is pressed. In L mode, the changes are transferred after the execution of an EDT statement. Only the modified records are written back to the file. On a write operation, the ISAM key is formed from the line number. It is not possible to renumber lines when performing real processing.

A file's character set cannot be modified during real processing. The @CODENAME statement is rejected with the message EDT5452.

Line numbers that are longer than the key length cannot be generated during real processing. The corresponding statements are rejected with an error message in the same way as when an excessively large increment is set.

Errors in the file (e.g. non-numerical keys, duplicate keys, illegal byte sequences etc.) may sometimes not be detected until the corresponding records are to be read. Not all the records in the file are read in when the file is opened. Any errors that are subsequently detected are then reported by the command which was executing when they occurred. In such cases, a file that has been opened for real processing is automatically closed.

6.7.3 Closing

A file that has been opened for real processing is closed with @CLOSE. It is also closed if one of the statements @HALT, @EXEC, @LOAD, @OPEN (format 2), @DELETE (format 2) or @DROP is entered.

The work file is deleted after the close operation.

6.8 Reading and writing POSIX files with the old statements

The old statements @XOPEN, @XCOPY and @XWRITE can still be used to access POSIX files. However, users are recommended only to use the new statements (see section “Reading and writing all supported file types” on page 138).

6.8.1 Reading

A POSIX file can be read into the current work file for processing using the @XOPEN statement. The file remains open until it is closed with @CLOSE.

The @XCOPY statement can also be used to read a POSIX file into the current work file. After being read in, the file is closed again.

6.8.2 Writing

The current work file can be written back to a POSIX file with @CLOSE provided that this has been previously read in with @XOPEN or if a new POSIX file has first been created with @XOPEN, MODE=NEW. After the write operation, the file is closed and the work file is deleted.

The current work file can be written to a new POSIX file with @XWRITE. The work file is retained after writing.

The current work file can be written to an existing POSIX file with @XWRITE. This completely replaces the old file content. The work file is retained after writing.

@XWRITE can be used to write the current work file back to a POSIX file which has been opened with @XOPEN. The file remains open after writing and the work file is retained.

6.9 File catalogs

The @SHOW statement (format 1) can be used to output lists of files from the BS2000 catalog, from a POSIX directory or from a library.

The @DELETE statement (format 3) can be used within EDT to delete files from the BS2000 catalog or from a POSIX directory or to delete elements from a library.

If the statements refer to library elements then all element types are permitted.

BS2000 file catalogs can also still be processed using the old statements. The @FSTAT statement can be used to output lists of BS2000 files. The @UNSAVE statement can be used to delete files from the BS2000 catalog from within EDT.

6.10 System files

EDT makes it possible to read from SYSDTA and write to SYSOUT and SYSLST.

The handling of character sets when accessing system files is described in section "Character sets" on page 47. The system files SYSLST and SYSOUT should only be assigned to files or library elements with a Unicode character set if it is certain that only EDT sends output to these files. Otherwise files containing invalid characters could be created since other system components do not usually take account of the character set assigned to SYSLST or SYSOUT.

6.10.1 The SYSDTA system file

EDT reads from SYSDTA in the following cases:

– When reading lines in L mode if @EDIT ONLY is set or in batch mode. If the line represents an EDT statement then it is executed immediately. Otherwise, the line is stored in the current work file.

– When reading with the statement @CREATE ... READ without a prompt or in batch mode.

The records that are to be read from SYSDTA may have a maximum length of 32763 bytes (RDATA permits an input area of 32767 bytes, where the first 4 bytes constitute the record length field).

When EDT is initialized, the SYSDTA character set is determined.
This is used when reading from SYSDTA. If the assignment to SYSDTA changes then the character set is determined again and reading is subsequently performed with the new character set.

6.10.2 The SYSOUT system file

In interactive mode, the following EDT output is written to SYSOUT:

1. Both in F mode and in L mode, the output from the statements @COMPARE (format 1), @LIMITS, @ON COLUMN, @SEQUENCE CHECK, @TABS VALUES is sent to SYSOUT.
2. In L mode only, the output from the statements @COMPARE (format 2), @FSTAT, @PROC (format 2), @SHOW, @STAJV, @STATUS is written to SYSOUT.
3. In L mode only, EDT error messages are written to SYSOUT.
4. Logging output resulting from the statements @CHECK (L mode only), @DO PRINT, @EDIT PRINT, @INPUT PRINT is written to SYSOUT.
   In this case, work file records or EDT statements (sometimes with additional information) are output.
5. Both in F mode and in L mode, the output from the statements @ON PRINT and @PRINT (without the V operand) is written to SYSOUT. In this case, the work file records are output together with line numbers.

In batch mode, only the messages concerning normal or abnormal termination (e.g. EDT8000) are sent to SYSOUT unless job switch 8 is set.

If an error that cannot be corrected occurs when writing to SYSOUT then EDT is terminated.

The first character in each record when output is written to SYSOUT is a line feed character. If SYSOUT is assigned to a terminal then it is not displayed. If SYSOUT is assigned to a file then it becomes a component of the file. If a non-existent file is assigned, then the system indicates in the catalog that the file contains EBCDIC control characters. However, EDT does not evaluate the catalog entry but always generates EBCDIC line feed characters or the control characters which correspond to these EBCDIC characters in the character set which is assigned to SYSOUT.
If the user wants to print the file then it is possible to evaluate these line feed characters. If output is sent to SYSOUT then the same line feed characters are used as in the case of SYSLST (see section “The SYSLST system file” on page 152).

The length of output to SYSOUT is restricted (2032 bytes including the record length field and line feed characters on output to files and 32767 bytes on output to the terminal).
If the output is longer than this then the record to be output is subdivided into sections of maximum 2027 bytes in the case of output to a file and 32762 bytes in the case of output to a terminal and the record is then output in several sections.
If SYSOUT is using a Unicode character set, this ensures that the line feed always takes place at a character boundary.

EDT sends output to SYSOUT in the assigned character set which is determined using the BS2000 macro GCCSN except in the case of the terminal. Output to the terminal is always sent in the specified communications character set. If the assignment to SYSOUT changes then the character set is determined again and writing is subsequently performed with the new character set. If this character set is *NONE then EDF03IRV is used. If the output contains characters which cannot be displayed in the target character set then the substitute character defined with @PAR SUBSTITUTION-CHARACTER is used. If no such character is defined, a blank is inserted.

When assigning a character set to SYSOUT it is always important to consider what components send their output there since not all system components take account of the SYSOUT character set correctly.
For example, in interactive mode, EDT output generated with WRTRD is sent to the terminal in the communications character set.
In this case, if SYSOUT is assigned to a file, then the system also writes the data present in the communications character set (e.g. UTFE) and any consequent user input (also in this character set) to this file without taking account of the file's character set (with which SYSOUT has been harmonized). This may cause problems if the file uses a different character set.
It is therefore currently advisable only to use EBCDIC character sets as the SYSOUT character set and to avoid redirecting SYSOUT to a file in interactive mode if at all possible.

If screen monitoring is active and SYSOUT is assigned to a terminal then output can be interrupted by pressing K2. If the EDT session is continued with /RESUME-PROGRAM or /INFORM-PROGRAM then the output is aborted and one or more messages are issued.

6.10.3 The SYSLST system file

In both interactive and batch mode, the following EDT output is written to SYSLST:

1. Output from the statements @LIST (without I operand) and @PAGE is written to SYSLST. In this case, the work file records are output together with line numbers.
2. Logging output resulting from the @LOG statement is written to SYSLST. In this case, work file records or EDT statements are output.

In batch mode, the following EDT output is also written to SYSLST unless job switch 8 is set:

3. Output from the statements @COMPARE, @FSTAT, @LIMITS, @ON COLUMN, @PROC (format 2), @SEQUENCE CHECK, @SHOW, @STAJV, @STATUS, @TABS VALUES is written to SYSLST.
4. EDT error messages are written to SYSLST.
5. Logging output resulting from the statements @CHECK, @DO PRINT, @EDIT PRINT, @INPUT PRINT is written to SYSLST. In this case, work file records or EDT statements (sometimes with additional information) are output.
6. Output from the statements @ON PRINT and @PRINT is written to SYSLST. In this case, the work file records are output together with line numbers.

If job switch 8 is set then EDT writes this output to SYSOUT in batch mode. Although this is not explicitly formulated in the statement description, it applies even if only '*is output to* SYSLST *in batch mode'* is stated.

If it is not possible to write to SYSLST in batch mode then the output is aborted and the error message EDT5498 is output at SYSOUT.

The first character in each record when output is written to SYSLST is a line feed character. If a non-existent file is assigned, then the system indicates in the catalog that the file contains EBCDIC control characters. However, EDT does not evaluate the catalog entry but always generates EBCDIC line feed characters or the control characters which correspond to these EBCDIC characters in the character set which is assigned to SYSLST. They are evaluated if SYSLST is printed during task termination. If a file assigned to SYSLST is printed then the user can trigger this evaluation.
If SYSLST has a Unicode character set then the control characters are converted accordingly. It is therefore possible to process the file in EDT. When the file is printed, the control characters are converted back again by the BS2000 SPOOL subsystem.

The line feed characters generated by EDT are presented in detail in the description of the @LIST statement as is the treatment of Unicode files with line feed characters during print operations.

A number of statements generate additional line feeds at the start of their output or at the start of a new section.
The line feed is usually omitted if the output occurs at the start if a page. The page size for SYSLST is set with the @PAGE and @LIST statements and applies to all output to SYSLST. If the SYSLST assignment is modified during the EDT session then EDT assumes that this has occurred at the start of a page and restarts its line count accordingly.

Output to SYSLST is usually limited to a maximum of 132 characters (plus line feed characters). If job switch 6 is set then the maximum length is 160 characters.
If the output is longer then it is split accordingly and output over several different sections. Line feeds are always performed at character boundaries.

Output to SYSLST is performed in the assigned character set which is determined using the BS2000 macro GCCSN. If the assignment to SYSLST changes then the character set is determined again and writing is subsequently performed with the new character set. If this character set is *NONE then EDF041 is used. If the output contains characters which cannot be displayed in the target character set then the substitute character defined with @PAR SUBSTITUTION-CHARACTER is used. If no such character is defined, a blank is inserted.

6.10.4 The system files SYSLST01 .. SYSLST99

EDT logging output resulting from the @LOG statement can be written to the files SYSLST01 to SYSLST99 in both interactive and batch mode.

It is only possible to send output to these system files if they are associated with a file, a library element or an S variable.

The first character in each record when output is written to SYSLST01 to SYSLST99 is a line feed character.
If a non-existent file is assigned, then the system indicates in the catalog that the file contains EBCDIC control characters. However, EDT does not evaluate the catalog entry but always generates EBCDIC line feed characters or the control characters which correspond to these EBCDIC characters in the character set which is assigned to the relevant SYSLSTnn file. If the user wants to print the file then it is possible to evaluate these line feed characters.
If output is sent to SYSLST01 to SYSLST99 then EDT only uses a small number of line feed characters, and in particular the page size is not monitored as in the case of SYSLST.

Output to SYSLST01 to SYSLST99 is usually limited to a maximum of 132 characters (plus line feed characters). If job switch 6 is set then the maximum length is 160 characters. If the output is longer then it is split accordingly and output over several different sections. Line feeds are always performed at character boundaries.

Output to SYSLST01 to SYSLST99 is performed in the assigned character set which is determined using the BS2000 macro GCCSN. If the assignment changes then the character set is determined again and writing is subsequently performed with the new character set. If this character set is *NONE then EDF03IRV is used. If the output contains characters which cannot be displayed in the target character set then the substitute character defined with @PAR SUBSTITUTION-CHARACTER is used. If no such character is defined, a blank is inserted.

7 Description of the statements

This section explains the notational conventions used in the detailed descriptions of the statement and statement codes, the basic structure of the descriptions and the operand types used in the various statements.

7.1 Metasyntax

The following metasyntax and typographic conventions are used for the formal presentation of the statements.

Formal representation	Explanation	Examples
UPPERCASE **and** special characters	Uppercase characters and special characters designate constants or keywords which the user must enter in exactly the presented form.	UPDATE, OVERWRITE
UPPERCASE semibold	Semibold uppercase letters indicate the short form of the keywords. Any input between the short form and the long form is permitted.	**@LOW**ER The user may enter: @LOW, @LOWE or @LOWER
lowercase	Lowercase letters describe variable operands which the user must replace with current values during input.	@GOTO line The user may enter, for example: @GOTO 3
{ }	Braces enclose a number of alternatives, i.e. one of the entries must be selected.	@**LOW**ER { **ON** / **OF**F } The user may enter: @LOWER ON or @LOWER OFF

|	| separates alternatives when these are not located above but next to one another.	**@LOW**ER {**ON** | **OF**F}
. .	**. .** designates alternatives which are not listed individually but have to be selected from a continuous range.	1**. .**22 The user may enter a value between 1 and 22. $1**. .** $22 The user may enter a symbol between `$1` and `$22`.
[ ]	Specifications in square brackets are optional and may be entered if the user wishes.	
[,...]	This construction with three dots indicates the possible repetition of the preceding syntactic unit. A comma must be entered as a separator between the repetitions.	`line` **[,...]** The user may enter, for example: `1, 3, 7` or `10`.
Underscore	Value used by EDT if none of the possible alternatives is specified. If, in such a case, none of the alternatives is identified as the default value then refer to the detailed description to determine EDT's behavior.	**@LOW**ER [{ **ON** / **OF**F }] The entries @LOWER and @LOWER ON have the same effect.

7.2 Statement syntax

This section explains the underlying concepts during the syntactic analysis of EDT statements.

Because the two types of input are handled differently, when ever input is received EDT must decide whether this is data input or an EDT statement (multiple EDT statements are also possible. In F mode, these are separated by semicolons (;),and in L mode by LZE characters provided that BLOCK mode has been activated).

In F mode, EDT makes this decision on the basis of the location of the input. EDT always interprets input in the statement line as an EDT statement and input in the data window as data input. It interprets input in the statement code column as statement codes. The same principle applies to the subroutine interface where EDT statements and data have to be entered in separate input areas provided for the purpose.

In contrast, in L mode all the input is made in a line. To make it possible to distinguish between data input and EDT statements in L mode, all EDT statements entered in L mode must start with the EDT statement symbol (default value: @).
In F mode and at the subroutine interface, the EDT statement symbol may be omitted since there is no danger of confusion with data input.

In L mode, the following special characteristics should also be noted.
If an entry starts with two EDT statement symbols (@@, where one or more blanks may be located before and after the first @), EDT interprets the input as data input and the second EDT statement symbol is considered to be the first character of the data input. EDT removes all the characters (first EDT statement symbol and all the blanks) which occur before the second EDT statement symbol. In L mode, it is therefore easy to write lines containing EDT statements in EDT procedures (if an input starts with a single EDT statement symbol then it is immediately executed as an EDT statement and is not written to a line). In L mode, therefore, data input is only interpreted as a statement if the first character which is not a blank is the EDT statement symbol and the first character that is not a blank following the EDT statement symbol is not the EDT statement symbol.

The above applies equally to user statement symbols. If input starts with two **identical** user statement symbols then this input is interpreted as data input and the second user statement symbol is considered to be the first character in the data input. In contrast, if the input starts with one user statement symbol or two different user statement symbols then everything after the first user statement symbol is interpreted and executed as a user statement (blanks are skipped).

Some EDT statements (e.g. @SET, format 6) possess the operand text (see section “Operand syntax” on page 164). This text operand which EDT handles as a separate input, may in turn take the form of either an EDT statement or a data input. EDT decides which interpretation is correct on the basis of the rules used in L mode.

If EDT identifies the input as an EDT statement then, in case the input consists of multiple EDT statements, it first isolates the first non-processed EDT statement in the input. In F mode, the semicolon is used as the separator and semicolons are not taken into account when literals are broken down. In L mode, the separator is the LZE character. The (separated) EDT statement is then copied to two internal buffers. One of the two buffers then contains the EDT statement as it was input while, in the other, it is converted into uppercase to simplify the recognition of the statement name and operands.

EDT then attempts to determine the statement name. If it is possible to identify the statement name and if it corresponds to an EDT statement with an indirect operand specification (see section "Indirect operand specification" on page 161) then the operands are now entered in the two internal buffers and in one of these, the input is again converted into uppercase.

A syntax check of the EDT statement is then performed.
During the analysis of EDT statements, EDT accesses the originally entered statement for those sections in which the distinction between uppercase and lowercase is relevant, e.g. literals.
Unicode substitute representations in EDT statements are only interpreted inside of literals (except in @DO and @PARAMS).
Furthermore, no tabulator expansion is performed in EDT statements. If there are no syntax errors, the EDT statement is now executed and the originally entered EDT statement is then written to the statement buffer (indirect operands are not resolved in this buffer).

An EDT statement begins with a statement name (e.g. @OPEN, @COPY, @WRITE) which may be followed by one or more operands. In the case of some EDT statements, a comment is permitted after the operands. One or more blanks are permitted (but not necessary) before the EDT statement symbol and between the EDT statement symbol and the statement name.

If an EDT statement possesses operands then these follow the statement name, possibly separated by one or more blanks. The operands must be entered in the predefined order. Any number of blanks may be entered before or after each operand. Some operands must always be specified whereas others are optional.

If optional operands are omitted then default values are assumed for these operands. The syntax description for each of the statements indicates which operands are optional and which are not and what the default values for omitted operands are.

The blanks between the statement name and the operands or between the individual operands themselves may be omitted. However, they must be entered if it would otherwise not be possible to distinguish between the statement name and the operand or between two successive operands.

Example

@SYMBOLS='?' is incorrect; the correct form is @SYMBOL S='?', since @SYMBOL is a legal abbreviation of the statement @SYMBOLS.

Users are generally not advised to omit the blanks between the statement name and the operands or between the individual operands since this may sometimes make statements very difficult to understand.

Alongside the direct entry of operands as described above, they can also be specified indirectly by means of string variables. This means that it is possible, for example, not to specify operands until runtime, thus permitting much greater flexibility, in particular in EDT procedures. This method of specifying operands is described in more detail in section "Indirect operand specification" on page 161.

In the case of some EDT statements, a comment is permitted after the operands if present. The syntax description of each statement indicates whether or not it may be accompanied by a comment.

Most EDT statements can be abbreviated. This is usually achieved by omitting one or more characters at the end of the statement name. In some cases, however, there are also abbreviations which do not result from the omission of characters. The @BLOCK statement, for example, can be abbreviated as @BK, the @QUOTE statement can be abbreviated as @QE and the @SETF statement in F mode can be reduced to the character #. In the @SEARCH-OPTIONS statement, it is also possible to omit certain characters in the middle of the statement name with the result, for example, that @SEA or @SEA-OPTIONS constitute valid abbreviations for these statements. The @SET statement, in which the statement name can be entirely omitted, is an exception here. However, if @SET (format 6) is used in F mode or at the subroutine interface then the statement symbol must be specified to remove ambiguity. The minimum portion of the statement name that must be present so that EDT can recognize the statement without ambiguity is indicated in bold print in the syntax diagrams.

Starting with the shortest possible abbreviation, EDT attempts to uniquely identify the statement name. If it is successful, any further characters in the statement name are skipped. This operation continues until EDT has processed the complete statement name or a character which does not correspond to the statement name is detected (this may also be a blank). The first character other than a blank that does not correspond to the statement name is interpreted as the first character of the operand section.

In the case of three pairs of statements (@DELETE/@DELIMIT, @PAR/@PARAMS and @UNSAVE/@UPDATE), the analysis of the statement name is not sufficient for a unique identification of the statement since the shortest possible abbreviations are identical (@D, @PAR and @U respectively). In these three cases, the first character of the operand section is used for differentiation. In the case of the @DELIMIT statement, for example, the operand section starts with the character =, whereas this character does not occur in the operand section of the @DELETE statement (in the case of the @PARAMS, statement & is the first character in the operand section and in the @UNSAVE statement, it is the character ' . Neither of these characters occurs in the operand section of the other statement in the pair).

Example

Consider the input @DEL& (blanks between the statement name and the operand section can be omitted). The abbreviations @DIALOG, @DO and @DR (for @DROP) do not match. The only remaining abbreviation is @D which may stand for the statement @DELETE or for the statement @DELIMIT. However, the @DELIMIT statement can be excluded because the character = does not occur in the remainder of the statement. The statement name is therefore @DELETE. The characters E and L in the input are now skipped since they match the corresponding characters in the statement name. However, the character & does not match the corresponding character (E) in the statement name and is therefore the first (and in this case the only) character in the operand section.

The above description of the procedure employed when analyzing statements also makes it clear why error messages with no immediately obvious cause are sometimes output. If no statement can be identified in the input, the message EDT3101 (illegal statement) is output (e.g. @XD). However, there are a large number of situations in which the message EDT3002 (operand error) is output even though the message EDT3101 would be expected. Let us assume that in the above example, @DDL& is accidentally entered instead of @DEL&. Exactly as in the above statement analysis, EDT would come to the conclusion that the intended statement is @DELETE (@DELIMIT is not possible because there is no =). However, the very next character after @D in the input differs from the corresponding character in the statement name @DELETE. Therefore, everything as of this character is considered to belong to the operand section (DL&). However, the string DL& cannot be interpreted as a permitted operand for any of the three formats of the @DELETE statement and the message EDT3002 is therefore output.

Notes

– The delimiter characters for literals (by default, the characters ' and ") can be redefined using the @QUOTE statement. However, in the @DO and @PARAMS statements, the character ' is always used as the delimiter for literals irrespective of any redefinition performed using the @QUOTE statement.

– In @DO procedures, statements are not analyzed until the procedure parameters have been substituted. Only then is any indirect operand specification resolved.

– It is not permissible to include blanks in the keywords of statements or append comments at the end of statements (comments are only permitted at the end of a statement if this is explicitly indicated in the statement description).

– The statement @SYNTAX TESTMODE=ON activates test mode. In this case, with only a few exceptions, EDT statements are not executed but simply subjected to a syntax check in L mode. This makes it possible, for example, to make sure that EDT procedures are suitable for execution before running them (for more detailed information, see the @SYNTAX statement, page 531).

7.2.1 Indirect operand specification

In the case of indirect operand specification, all the operands that the user wants to specify in the corresponding EDT statement are stored in a string variable (#S00..#S20) before the EDT statement is executed. In the EDT statement itself, the indirect operand specification is introduced by the & character after the statement name and separated from it by one or more blanks. Thee & character must be immediately followed by the name of the string variable which contains the operands for the EDT statement. The statement may not contain any other characters (apart from blanks).

Once the statement name has been identified, the remainder of the statement (the & character followed by the name of the string variable) is replaced by the content of the string variable and the operands it contains are evaluated.

If logging is active (e.g. @LOG ALL or @LOG COMMANDS) then the statement generated by this substitution is output together with the original input.
If the line causing an error message is output then only the original input is specified.

If the length of the statement name together with the substitution of the string variable exceeds 32768 then processing is rejected with error message EDT5485.

The text operand in the statements @+, @-, @IF (format 1) and @SET (format 6) may itself be an EDT statement. Indirect operand specification is also possible in these EDT statements specified in the text operand.

No indirect operand specification is permitted in the statements @: (redefine the EDT statement symbol), @+, @- and value assignments to EDT variables with the @SET statement (the statement name @SET is omitted). This also applies to the @PARAMS statement since all its operands start with the character &.

Caution

If indirect operand specifications are used for the @IF and @SET (format 6) statements then endless loops may occur in EDT if the text operand in these statements itself contains an @IF or @SET (format 6) statement, e.g.:

```
1.     @SET #S1='1: @SET &#S1' ---------------------------------------- (1)
2.     @SET &#S1 ------------------------------------------------------- (2)
```

or:

```
1.     @SET #S1='1: @IF &#S2' ----------------------------------------- (1)
2.     @SET #S2='NO ERRORS: @SET &#S1' ------------------------------- (1)
3.     @SET &#S1 ------------------------------------------------------- (2)
```

(1) The string variables #S1 and #S2 are filled with suitable content prior to indirect operand specification.

(2) Once this statement has been issued with indirect operands, EDT enters an endless loop.

If additional different string variables are used, it is possible to construct statement sequences of any required level of complexity which may also cause EDT to enter an endless loop.

7.3 Structure of the statement descriptions

The detailed statement descriptions have a uniform structure:

1. Description of the function of the statement
2. Formal statement syntax
3. Detailed description of the operands
4. Description of the statement's special characteristics and limitations as well as notes on its use
5. Examples

Every statement description starts with a general description of the function of the statement. This is immediately followed by a formal description of the statement's syntax in the following form:

Operation	**Operands**	**Modes**
Operation	Operands	

The *Operation* field contains the name of the statement written out in full and the maximum permitted abbreviation of the application name is highlighted in semibold type. In the case of all the statements which have to be introduced with the statement symbol in L mode or which may be introduced by the statement symbol in F mode or at the subroutine interface, the default statement symbol @ is also specified. In the case of statements which may not be introduced by the statement symbol, it is omitted. In the case of a very few statements, two possible statement names are specified. It is possible to use either of these. Any other special characteristics which have to be taken into account when using the statement name or one of its abbreviations are explained in the following text section of the statement description.

The *Operands* field contains a formal syntactic description of the operands permitted for the statement as if they were specified directly in a statement line. This also corresponds to the syntax that must be observed when operand specification is made by means of a string

variable when the statement is to be called with indirect operand specification. Otherwise, there is no discussion of indirect operand specification itself in the statement descriptions (see section “Statement syntax” on page 157). However, if indirect operand specification is prohibited then this is mentioned in the text section of the statement description.

The naming of variable operands also identifies the operand type and thus describes the syntax of the values that can be used for it. For a description of all the operand types, see the section “Operand syntax” on page 164.

The *Modes* field lists the EDT operating modes in which the statement may be used. The following specifications are possible:

F mode — The statement may be used in F mode. If the field does not contain any other entries then the statement may only be used in F mode. The following text section of the statement description may indicate any other special characteristics relating to the use of the statement in F mode.

L mode — The statement may be used in L mode and consequently also in procedures. If the field does not contain any other entries then the statement may not be used in F mode. The following text section of the statement description may indicate any other special characteristics relating to the use of the statement in L mode.

@PROC — The statement is only permitted in procedures. It is not possible to use them in L mode outside of procedures and they may also not be used in F mode.

The formal description of the statement syntax is followed by a detailed description of the individual operands, usually in the order in which they occur. The description presents not only the function of the operand in question but also any special semantic considerations relating to the operands, the default values if operands are omitted and interactions between operands. However, the syntax of variable operands is not described. This can be found in the section “Operand syntax” on page 164.

The operand description is usually followed by a text section in which further special characteristics and restrictions relating to the current statement are explained. This section also contains any special comments concerning use.

Most of the statement descriptions end with one or more examples which again demonstrate the special characteristics of the statement. Unless indicated otherwise, all the examples assume the EDT default settings.

7.4 Operand syntax

This section contains the precise syntactic definition of the various variable operands which occur in EDT statements. An operand's name always makes it possible to identify its particular syntactic definition. Any special semantic considerations or restrictions within the context of the particular statement in question are indicated in the operand descriptions in the detailed statement descriptions.

All the operand types are defined in the following sections. These operand types are used as operand names in the detailed statement descriptions. If syntactically equivalent operands occur at various positions in a statement description then they are differentiated by appending a sequential number. A definition is provided only for the basic names of any such operands.

The following description of the operand syntax is subdivided into thematically linked sections. Within the sections, the descriptions are organized in such a way that, if possible, each operand type is defined before it is used for the first time. The following alphabetically ordered overview will help readers find the particular definitions they require.

Operand	Short description	Page
fraction	Part of a line number (after the decimal point)	171
freetype	Free type name of a library element	181
hd	Hexadecimal digit	166
hex	Sequence of hexadecimal digits	171
hpos	Relative horizontal positioning statement	185
inc	Increment for line numbers	177
int	Integer	171
intex	Integer expression	172
ivar	Integer variable	169
line	Line number specified directly or as an expression	178
lines	Contiguous range of line numbers	178
linkname	Link name for files or job variables	182
lnum	Directly specified line number	177
loopsymb	Loop counter	168
lsym	Symbolically specified line number	177
lvar	Line number variable	169
m	Record mark	185
message	Any message text	173
modlib	Library dynamically loaded from the module	183
n	Unsigned integer	171
name	String of maximum eight characters	173
op	Mathematical operator + or -	168
param	Parameter in @DO procedures	184
path	Path name of a DMS file or job variable	183
procnr	Name of a work file	185
progname	Name of a program	182
rangesymb	Range symbol	168
rel	Relation in an @IF statement	168
search	Search term in an @ON statement	175
spec	Special character	167
str	Quoted sequence of characters	173

Operand	Short description	Page
strchar	Quoted individual characters	174
strspec	Quoted individual special character	174
string	Directly or indirectly specified string	175
svar	String variable	169
svarex	Indirect specification of a string variable	170
svars	Contiguous range of string variables	170
text	Follow-up input in L mode statements	175
unicode	UTF16 code of a character (4 hexadecimal digits)	167
ver	Version number of a cataloged DMS file	183
vers	Version number of a library element	183
vpos	Relative vertical positioning statement	185
vpos–op	Vertical positioning operand	185
xpath	Path name of a POSIX file	184

7.4.1 Characters and symbols

This section contains descriptions of the elementary operand types as well as operand types which are only needed for the definition of other operand types but which do not themselves occur as real operands in any of the statements.

Operand	Definition
binary	0 | 1

The digit 0 or 1.

Operand	Definition
dd	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Decimal digit.

Operand	Definition
hd	dd | A | B | C | D | E | F | a | b | c | d | e | f

Hexadecimal digit.

Operand	Definition
spec	! " # $ % & ’ ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ‘ { | } ~

A special character from the group of characters specified above (see also section “Character set in a statement” on page 58).

Operand	Definition
unicode	hd hd hd hd

Sequence of precisely four hexadecimal digits which specify a character's `UTF16` code.

Operand	Definition
escsymb	spec

The current escape character which introduces a Unicode substitute representation `escseq`. It can be defined using the statement @PAR ESCAPE-CHARACTER. By default, no escape character is assigned.

Operand	Definition
escseq	escsymb U unicode

Substitute representation for a Unicode character. The sequence of hexadecimal digits for the unicode operand type must correspond to the character's `UTF16` coding. If the user has defined the escape character `%` with @PAR ESCAPE-CHARACTER=‘%‘ then, for example, `%U20AC` would be a valid Unicode corresponding to the character €.

Operand	Definition
char	Any character

Any character.

The group of available characters depends on the employed character set on the one hand, and on the input source on the other. Thus it is not possible to input any character via the keyboard even if the terminal is able to display this character. In the case of characters which cannot be entered directly, the operand type `char*` permits a Unicode substitute representation.

Operand	Definition
char*	char | escseq

Any character which is specified directly or in its `UTF16` coding in the form of a Unicode substitute representation (see also section "Substitute character representation in Unicode" on page 52).

Operand	Definition
rangesymb	spec

The current range symbol. This can be modified using the @RANGE statement. By default, this is the character `&`.

Operand	Definition
loopsymb	spec

The current loop counter which is defined in the @DO statement and which can be used in the same way as a line number variable in the called @DO procedure.

Operand	Definition
op	+ | -

One of the mathematical operators + or −.

Operand	Definition
rel	GT | LT | GE | LE | EQ | NE | > | < | >= | <= | = | <>

Character representing a relation which can be queried using the @IF statement.

The 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) and NE or <> (not equal to) have their usual mathematical meanings.

7.4.2 Variables

This section contains the definition of integer variables, line number variables and string variables together with expressions which determine these variables. The variables may be specified in many EDT statements instead of explicitly defined numbers, line numbers or strings.

Operand	Definition
ivar	#I00..#I20

One of the integer variables #I00,#I01,...,#I20 (see section "EDT variables" on page 61). Leading zeros in the numerical part of the variable designation may be omitted. Integer variables can be used to store positive or negative integer values. The permitted range of values is between -2^{31} and $2^{31}-1$. If an integer variable is used in a statement instead of an explicitly specified number then different limits apply depending on the statement.

Operand	Definition
lvar	#L00..#L20

One of the line number variables #L00,#L01,...,#L20 (see section "EDT variables" on page 61). Leading zeros in the numerical part of the variable designation may be omitted. A line number variable may have a value between 0.0001 and 9999.9999. When EDT is started, all the line number variables have the invalid value 0.0. Permissible values must then be assigned to the line number variables before they are used.

Operand	Definition
svar	#S00..#S20

One of the string variables #S00,#S01...,#S20 (see section "EDT variables" on page 61). Leading zeros in the numerical part of the variable designation may be omitted. Every string variable is assigned a content (a string) and a character set. If a string variable is deleted

then it has a blank as content and the character set EDF041. This is also the preliminary default setting for all string variables. String variables may be used as work file line numbers in many EDT statements.

Operand	Definition
svarex	svar[op ivar] | svar[op nL]

Indirect specification of a string variable in the form of an expression which describes its position relative to a given string variable. The relative position of the desired sequence of string variables can be defined by means of the content of an integer variable or explicitly by specifying nL.

Examples

– If #I10 contains the value 5, then #S0+#I10 designates the variable #S5.

– If #I3 contains the value 7, then #S10–#I3 designates the variable #S3.

– The expression #S15–5L designates the variable #S10.

– The expression #S3+8L designates the variable #S11.

Operand	Definition
svars	svarex [[.] - [.] svarex]

A contiguous range of string variables

Note

The entry of dots before and/or after the range separator and the entry of leading zeros before a variable name are still supported in the previous form for reasons of compatibility but are no longer necessary.

The specification svarex1–svarex2 (e.g. #S1–#S10) has the same effect as svarex2–svarex1 (e.g. #S10–#S1). If a svarex operand is now specified, the range consists of only this one string variable.

Examples

– #S3 selects the string variable #S3.

– #S4–#S7 selects #S4,#S5,#S6 and #S7.

– #S2+1L–#S6–#I3 selects #S3 and #S4 if #I3 has the content 2.

7.4.3 Numbers

This section defines various numerical formats which are used in EDT. If certain formats define semantic rules for the specified number then these are described here.

Operand	Definition
n	dd | n dd

Unsigned integer.
The number of permitted digits depends on the statement in question. Therefore, `00005` does not necessarily have to be equal to `5`.

Operand	Definition
fraction	.dd | fraction dd

The part of a line number after the decimal point.
The values `.0001` to `.9999` are permitted.

Operand	Definition
hex	hd | hex hd

Sequence of hexadecimal digits.

Operand	Definition
int	n | op n | ivar

Integer value which can be specified either explicitly or via an integer variable, for example: `5,0,-23456` or `#I0,#I1,...#I20`.
Unlike the operand type `n`, only the numerical value is of significance in `int` and leading zeros do not cause any distinction.
When explicitly specified, the permitted range of values is between -2^{31} and $2^{31}-1$.

Operand	Definition
intex	int | op int | intex op int

Integer expression.

7.4.4 Strings

This section defines strings which are used with different semantics in EDT statements, for example in search statements, as comments or as special characters.

Strings without a defined delimiter (e.g. quotation mark) are first extracted during the syntax analysis without taking account of the associated semantic constraints.
In this case, all the characters from the first character that is not a blank up to an internally defined delimiter or the end of the statement are used (unless otherwise described for the operand type). EDT normally uses the blank, comma, equals sign and parentheses as a delimiter. Only when the operand has been extracted is it checked for length, characters used and permitted syntax.

Operand	Definition
chars	char | chars char

String.

Operand	Definition
chars*	char* | chars* char*

String which may contain not only characters from the associated character set but also substitute representations for Unicode characters. The substitute representation `escseq` (see `char*`) for Unicode characters can be used if a character cannot be entered directly at the keyboard or if Unicode characters are to be entered via files which are not themselves coded in Unicode (see also section “Substitute character representation in Unicode” on page 52).

If, for example, the character `%` is defined as `escsymb` using @PAR ESCAPE-CHARACTER=‘%‘ then `'Have you got a %U20ac?'` would be a valid operand of type `chars*`.

Operand	Definition
comment	chars

Any comment.

Operand	Definition
message	chars

Any text (up to the end of the statement, including blanks) which is passed to the calling program when EDT is called as a subroutine with @RETURN or @HALT.
The string may contain a maximum of 80 characters and only use printing characters from the `EDF03IRV` character set.

Operand	Definition
name	chars

String of maximum length of 8 characters corresponding to the SDF data type `<alphanum-name 1..8>`.

Operand	Definition
str	' [chars*] ' [*int] | B ' binary ' [*int] | X ' hex ' [*int]

Sequence of quoted characters specified either as characters from the associated character set or in their binary or hexadecimal coding. When displaying characters, the Unicode substitute representation `escseq` is also permitted. Whether or not a blank string is permitted in the character display depends on the statement in question and is set out in the associated description.

If `B` or `X` is used, then the binary or hexadecimal digit is always interpreted in the character set for the current work file (or in `EDF041` if the current work file does not have a character set) irrespective of what the employed command then does with the string.
If the string needs to contain a apostrophe then it is necessary to enter two apostrophe. The valid quote character can be modified using the @QUOTE statement.

The optional specification of *int is intended for the repetition of strings, e.g. 'ab'*3 is the equivalent of 'ababab'. Since the maximum length of a string is 32768, int must not exceed this value. If int has the value 0 or if the string that is to be repeated has the length 0 then the resulting string has the length 0.

Examples

– Specifying 'A''BC''D' generates the string A'BC'D.
– Specifying 'ABC'*5 generates the string ABCABCABCABCABC.
– Specifying X'C1F2'*4 generates the string A2A2A2A2 if EDF041 has been defined as the character set.
– Specifying B'11110000'*3 generates the string 000 if EDF041 has been defined as the character set.
– Specifying 'That is the %U0391 and %U03a9' generates the string 'That is the Α and Ω' if a Unicode character set has been defined and the character % has been declared for @PAR ESCAPE-CHARACTER.

Notes

– If an odd number of characters is used in a hexadecimal specification then the entry is left-filled with zeros. Thus X'F' is equivalent to X'0F' and X'A'*4 is equivalent to X'0A'*4.
– The same applies to binary representations if the number of binary characters is not a multiple of 8. Here again, the value is left-filled with zeros until the number of binary characters is a multiple of 8. Thus B'1' is the equivalent of B'00000001' and B'1111'*2 is the equivalent of B'00001111'*2.

Operand	Definition
strchar	str | U'unicode'

Individual character in quotes or in binary or hexadecimal coding or direct specification of UTF16 coding. The resulting string must have precisely the length 1.

Operand	Definition
strspec	str

Individual character in quotes or in binary or hexadecimal coding. The resulting string must have precisely the length 1 and come from the group of characters defined in spec.

Operand	Definition
string	str | line[:cols[,...] [:]] | svarex[:cols[,...] [:]]

A directly or indirectly specified string.

If `string` is specified indirectly via a string variable or if a line number is specified then EDT uses the content of the string variable or the content of the corresponding line as `string`. If no such line exists, an error message is output and the statement is rejected.

If only a portion of a line or a string variable is required as `string` then this can be defined by means of the appropriate column specifications. If column values which exceed the line length are specified then a corresponding number of blanks are used in their place.
For example, if line 6 contains the string `AB3CD6EF9` and `string` is specified as `6:1-3,9,8,9,8-9,5-7,30,1,30-32,1:`, then the corresponding string represented by this expression is `AB39F9F9D6E␣A␣␣␣A`.

If the `string` specification in a statement consists of a line number which is itself modified by the EDT statement then the original content of the line is used as the operand. If, for example, line 1 has the value `ABC` and the EDT statement is @CREATE1:1,'D'*3,1, then, after execution of the statement, line 1 has the value `ABCDDDABC`.

Operand	Definition
search	string

Analogous to the `string` operand type but, however, a *quotation mark (")* can be used instead of an *apostrophe (')* within the alternative `str`. Both characters can be redefined using the @QUOTE statement.

The `search` operand type is only used in the @ON statement in order to define the search term.
For an explanation of the meaning of the *apostrophe* and *quotation mark* characters in a search statement, see “Searching with @ON” on page 78.

Operand	Definition
text	chars*

Follow-up input in certain L mode statements.

The text is treated in the same way as input in L mode, i.e. it is either considered to be a statement and executed immediately or it is considered to be data input and inserted in the current work file at the position of the current line number. EDT is able to determine the nature of the input depending on whether the first characters other than blanks in the text consist of one, two or no statement symbols or user statement symbols (see section “Input in L mode” on page 126).

If the text consists of data input and contains Unicode substitute representations – `escseq` – then these are only interpreted as Unicode characters if the employed escape character has been defined with @PAR ESCAPE-CHARACTER and @PAR DATA-REPLACEMENT=ON has been set. If the text is a statement then Unicode substitute representations are only interpreted as Unicode characters inside literals. In this case, the interpretation is independent of the setting of @PAR DATA-REPLACEMENT.

7.4.5 Lines and line ranges

This section defines the various formats which can be used to address lines and line ranges in EDT statements.

Operand	Definition
lnum	n | fraction | n fraction

Line number.
Values between `0.0001` and `9999.9999` are permitted for `lnum`.

Operand	Definition
inc	lnum

Increment for line numbers.
Values between `0.0001` and `9999.9999` are permitted for `inc`.

Operand	Definition
lsym	lvar | * | % | $ | ? | loopsymb

Symbolically specified line number which is specified either as a line number variable or as one of the symbols explained below (see also section "Symbolic line numbers" on page 35).

`*` Current line number, i.e. the line number which EDT last wrote to the terminal as an acknowledgment in L mode. If the file is empty, `*` has the value `1`.

`%` Lowest line number in the file. If the file is empty, `%` has the value `1`.

`$` Highest line number in the file. If the file is empty or possesses only a single line then `$=%`.

`?` Line number of the first hit line resulting from a preceding @ON statement. The value when EDT starts is `0`. This can only be modified by a successful @ON statement. Following an @ON, the symbolic line number `?` therefore has the same value as `#LOO`.

The symbolic line numbers `*`,`%`,`$` and `?` always refer to the current work file even if they are used in a range specification for another work file or for an external file. Their values at any given time can be output using the @STATUS statement.

Operand	Definition
line	{ lsym [op inc] | lnum } [op { ivar | nL | lsym }]

The `line` operand can be used to specify line numbers directly or as an expression which describes their position relative to other line numbers.
If neither `ivar` nor `nL` occurs in the expression specified for `line` then the line number is calculated as an absolute value, i.e. the line number is determined by adding or subtracting the values of `lsym` and/or `lnum`.
If `ivar` or `nL` is specified then a logical line number is determined, i.e. the number of existing lines specified by means of `ivar` or `nL` are skipped starting from an absolute value, independently of the increment used for line numbering.

In the expression `nL`, `n` may not have the value `0`. However, it is possible to store this value in an integer variable. It is only possible to assign a logically determined line number if a corresponding line actually exists. Otherwise an error message is output.

Examples
- `17.1` addresses the required line directly and absolutely.
- If `*=50.1` and `%=1.0000`, then `*+3.5-%` addresses the line `52.6000` absolutely.
- If `%=1.0000` and `#I15=6`, then `%+#I15` addresses the 6th logical line after line number `1.0000` (this is not necessarily line `7.0000`).
- If `%=1.000`, then `%+2L` addresses the 2nd logical line after `1.0000`.
- If `%=1.0000` and `*=3.0000`, then `%+*` addresses line `4.0000`.
- If `*=50.1` and `#I5=1`, then `*+3.5+#I5` addresses the line which logically follows line `53.6000` (this is not necessarily line `53.7000`).
- If `*=50.1000`, then `*+3.5+6L` addresses the 6th logical line after `53.6000`.

Operand	Definition
lines	rangesymb | line[[.] - [.] line]

A contiguous line range.

Specifying `line1-line2` (e.g. `1-10`) has the same effect as `line2-line1 (10-1)`. If only one `line` operand is specified then the line range consists of only this one line.

The rangesymb operand represents the range symbol which can be declared by means of the @RANGE statement. The default setting is the character '&' and the range 0.0001-9999.9999.
Since the minus sign can both be used as a symbol for defining range limits and occur with its arithmetical meaning in the line expression, ambiguous cases may arise. The following conventions help overcome this problem:

– If the first range limit ends with lsym, then the correct notation is lsym.-line.
– If the second range limit starts with lsym, then the correct notation is line-.lsym ... or line-.lsym op ...

Specifying . (period) makes it clear that the expression describes a range and not a difference. The figure 0 can be entered instead of a period. Unlike in svars, the period may only be entered in the specified cases here.

Examples

– 1-10 specifies the lines 1 to 10.
– %.-5 selects the range from the 1st line in the file through to line 5.
– %+5L-.$-10L selects the range from the 6th line in the file through to the 10th line before the end of the file.
– %.-$ specifies the entire file.
– *+2.1-?.-.%+5L expresses the range *+2.1-? through to the 6th line of the file.
– #L1.-#L2 designates the range from #L1 to #L2.
– 12.011 selects only the line 12.011.
– #L9 selects only the line whose number is stored in #L9 .

7.4.6 Columns and column ranges

This section defines the various formats which can be used to address columns and column ranges in EDT statements.

Operand	Definition
col	int

Column number which may have a value between `1` and `32768`. Nevertheless, some EDT statements demand a smaller `col` value.

Operand	Definition
cols	col[-col]

A contiguous column range.

The second `col` must not be smaller than the first. If it is not specified then the first `col` may either designate the specified column or the range from `col` through to the end of the line. Which of these applies can be found in the descriptions of the relevant statements.

If the second `col` is specified and is greater than the line length then the column range extends through to the end of the line in question. The procedure adopted if the first `col` is greater than the line length is specified in the descriptions of the relevant statements.

Operand	Definition
cols*	col[-col]

Column range in which the associated column number is specified relative to the end of the record. The rules specified for `cols` apply equivalently to `cols*`. However, the range `col1-col2` for each line must be replaced by (`linelength-col2+1`)-(`linelength-col1+1`). If this results in negative column numbers, the value `1` should be used.

7.4.7 File names and other system designations

This section defines special formats for strings which are used in EDT statements to designate external objects such as files or library elements.

Strings without a defined delimiter (e.g. apostrophe) are first extracted during the syntax analysis without taking account of the associated semantic constraints.
In this case, all the characters from the first character that is not a blank up to an internally defined delimiter or the end of the statement are used (unless otherwise described for the operand type).
EDT normally uses the blank, comma, equals sign and parentheses as a delimiter. Only when the operand has been extracted is it checked for length, characters used and permitted syntax.

Operand	Definition
entry	chars | .svar

Name of an entry point (`ENTRY`) or a `CSECT` section. This specification is case-sensitive. The name must not be longer than 32 characters and must comply with the BLS constraints for symbol names.

Operand	Definition
freetype	name

Free type name of a library element, specified as a string of 2 to 8 characters in length which may not begin with `$` or `SYS`.

Operand	Definition
eltype	S | M | R | C | P | J | D | X | H | L | U | F | *STD | freetype | .svar

Type of a library element.
Free type names can also be used to specify an element type. No check of the basic type is performed. In some statements, only text types are permitted.

Operand	Definition
elname	chars | .svar

Name of a library element which corresponds to the SDF data type <composed-name 1..64 with-under>.

Operand	Definition
progname	chars

Name of a program whose statements are to be subjected to a syntax check by EDT. The name must correspond to the SDF data type <structured-name 1..30>.

Operand	Definition
file	str

File name which can be specified by means of a printable, hexadecimal or binary representation.
The file name may consist of a maximum of 54 characters without wildcards or 80 characters with wildcards. The DMS constraints for file names must be taken into account.
The specification of wildcards is not permitted in all statements. These are the wildcards accepted in DMS, not the wildcards patterns declared in EDT.
Whether or not partial file name specifications are permitted again depends on the relevant statement and is set out in the corresponding description.
In some statements, it is also possible to specify '/' to indicate the use of a given link name. This is explained in the descriptions of the corresponding statements.

Operand	Definition
linkname	chars

Specifies a file or job variable via its link name.
The name must correspond to the SDF data type <filename 1..8 without-gen>.

Operand	Definition
path	chars | .svar

Path name of a file or a job variable which can be specified either directly or via a string variable.
The path name may consist of a maximum of 54 characters without wildcards or 80 characters with wildcards. The DMS constraints for file names must be taken into account.
The specification of wildcards is not permitted in all statements.
These are the wildcards accepted in DMS, not the wildcards patterns declared in EDT.
Whether or not partial file name specifications are permitted again depends on the relevant statement and is set out in the corresponding description.

Operand	Definition
modlib	path

Library containing a module that is to be loaded by EDT.
The name must correspond to the SDF data type `<filename 1..54 without-vers>`.

Operand	Definition
ver	* | int

Version number of a cataloged file.
This can be specified either as `*` or `int` where `int` stands for a number between 0 and 255.
For information on using version numbers when reading and writing files, see section “Version numbers” on page 141.

Operand	Definition
vers	chars | *STD

Version designation of a library element.
The designation must correspond to the SDF data type `<composed-name 1..24 with-under>`.

Operand	Definition
xpath	chars | .svar

String which specifies the name of a POSIX file.
The specification of the complete path name is permitted. If no complete path name is specified then the file is located in the current POSIX directory.
Blanks and commas in a name are only permitted if the name is specified in `svar`.
The path name must correspond to the SDF data type `<posix-pathname 1..1023>`.

7.4.8 Other

This section describes syntax elements which do not correspond to any of the categories described above.

Operand	Definition
formal	&id

Formal parameter of the form `&id` which is to be specified in the @PARAMS statement of a @DO procedure.
The remainder of the operand – `id` – is the name and may consist of 7 letters or digits. The first character must be a letter.
This operand is used for keyword and positional parameters.

Operand	Definition
param	' [chars*] ' | chars

Parameters which are passed to a procedure file for execution via @DO.
These consist of a freely defined string which must be quoted if a comma, a closing parenthesis or a Unicode character with substitute representation is to be passed as part of the parameter sequence.

In this case, each apostrophe that is to be passed in the parameter expression must be identified by duplicate apostrophes. The delimiting apostrophes can be redefined using @QUOTE.

Operand	Definition
procnr	int

Number of a work file.
Values between `0` and `22` are permitted. In some statements, the value `0` is not permitted.

Operand	Definition
m	dd | ivar

Record mark `1..9`.

Operand	Definition
hpos	>[n] | <[n] | <<

Relative horizontal positioning statement.

Operand	Definition
vpos	op n | vpos-op | vpos-op (m[,...])

Relative vertical positioning statement.

Operand	Definition
vpos-op	+ | - | ++ | --

Vertical positioning operand.

8 Statement overview

This overview presents the EDT statements ordered by topic and provides a brief description. For the precise operand syntax, a detailed functional description and notes on any restrictions or error messages, refer to the following alphabetically ordered description. If the same statement can be assigned to a number of different topic groups then it is listed more than once.

A corresponding comment indicates the statements which are no longer supported in EDT V17.0 Unicode mode.

8.1 EDT parameter settings

The following statements are used to modify the predefined EDT parameter settings and thus enable users to largely adapt the behavior and appearance of EDT to their own requirements.

@:	Defines a new statement symbol.	F mode L mode
@AUTOSAVE	Activates the automatic time-controlled saving of work files.	F mode L mode
@BLOCK **@BK**	Activates or deactivates EDT's blocked input/output mode (BLOCK mode).	F mode L mode
@CHECK (Format 1)	Activates the logging of all lines that are created or modified in a work file or a string variable by a statement. It also makes it possible to check the number of characters per line.	F mode L mode
@CHECK (Format 2)	Causes EDT to check whether the specified range in the current work file or range of string variables can be converted into the target character set without loss.	F mode L mode
@CODE	This statement is only supported in compatibility mode.	F mode L mode

@**CODEN**AME (Format 1)	Defines the character sets for work files and string variables.	F mode L mode
@**CODEN**AME (Format 2)	Specifies the communications character set which EDT uses in interactive mode in order to exchange data with the terminal.	F mode L mode
@**D**ELIMIT	Declares characters that act as delimiters when searches are performed with @ON.	F mode L mode
@**I**NPUT (Format 3)	Specifies how EDT is to interpret text input in L mode.	L mode
@**LOW**ER	Specifies whether or not EDT converts lowercase characters into uppercase when data and statements are input at the terminal.	F mode L mode
@**P-K**EYS	Loads the keyboard's programmable keys (P keys) with a default assignment predefined by EDT or displays the EDT predefined default assignment.	F mode L mode
@**PAR**	Specifies the EDT parameter settings. These settings control the screen display, behavior on input, default values for statements and the declaration of special meanings for certain characters.	F mode L mode
@**Q**UOTE @**QE**	Redefines the delimiter characters *apostrophe* and *quotation mark*.	F mode L mode
@**RA**NGE	Declares a symbol for a line range.	F mode L mode
@**SEA**RCH-OPTION	Makes presettings for searches using the @ON statement.	F mode L mode
@**SETS**W	User and job switches are set or reset.	F mode L mode
@**SYM**BOLS	Declares the wildcard symbols *asterisk* and *slash* for searches using placeholders. The `FILLER` operand declares a filler character.	F mode L mode
@**SYN**TAX	Defines the type of syntax check for input in L mode. It is also possible to activate or deactivate the test mode.	F mode L mode
@**T**ABS (Format 1)	Tabulator positions are defined for positioning with the hardware tabulator and the current values of these positions are output.	F mode L mode
@**T**ABS (Format 2)	Tabulator characters and tabulator positions are defined for positioning with software tabulators and the current values are output.	F mode L mode

@**V**DT	The screen format for F mode is set. The function no longer has any function in L mode and is supported for reasons of compatibility only.	F mode L mode
@**VT**CSET	On output to SYSOUT, specifies whether the line mode control characters which may be present in the lines of data that are to be output are transferred unchanged or are converted into smudge characters.	F mode L mode
@**E**DIT (Format 2)	In the interactive mode's L mode, this switches the input stream to terminal input. WRTRD is used for reading and the current line number is output as the prompt. If the statement is entered in F mode, operation first switches to L mode.	F mode L mode
@**E**DIT (Format 3)	In the interactive mode's L mode, this switches the input stream to input from SYSDTA. Reading is performed with RDATA. If the statement is entered in F mode, operation first switches to L mode.	F mode L mode
@**E**DIT (Format 4)	In F mode, switches between the full display of records and the display of a record section in the data window for the current work file.	F mode
@**HEX**	Activates or deactivates hexadecimal mode for the current work file.	F mode
@**I**NDEX	In F mode, activates or deactivates the line number display for the current work file in the relevant data window.	F mode
@**SC**ALE	In F mode, activates or deactivates the display of a column counter (horizontal ruler) for the current work file in the work window.	F mode
@**SPLIT**	In F mode, activates or deactivates the display of a second work window on the screen.	F mode
@**Z**ERO- RECORDS	This statement is now only supported in compatibility mode.	F mode L mode

8.2 File processing

The file processing statements provide a uniform interface for all the file types supported by EDT (DMS files, library elements and POSIX files).
These statements therefore represent the preferred way of processing files. The old, non-uniform statements for the file types are now only supported for reasons of compatibility.

@**CL**OSE	Causes the current work file to be written back to disk or tape, opened SAM, ISAM or POSIX files or library elements to be closed and the work file to be deleted.	F mode L mode
@**C**OPY (Format 1)	Reads an existing SAM, ISAM or POSIX file or a library element in full into the current work file. The work file does not have to be empty when this is done. After being read in, the file or library element is closed again.	F mode L mode
@**D**ELETE (Format 3)	Deletes files or library elements.	F mode L mode
@**OP**EN (Format 1)	Opens an existing SAM, ISAM or POSIX file or a library element and reads it into the current work file or creates a new file and opens this for processing.	F mode L mode
@**SHOW** (Format 1)	Outputs a library's directory or a list of files from the BS2000 catalog or from a POSIX directory.	F mode L mode
@**W**RITE (Format 1)	Creates a new SAM, ISAM or POSIX file or a library element and writes the content of the current work file to the new file or overwrites an existing file with the content of the current work file or writes the content of the current work file back to a file opened using @OPEN (format 1). An opened file remains open when @WRITE is issued and the content of the work file is retained.	F mode L mode

8.3 Old statements for processing SAM and ISAM files

These statements for processing SAM and ISAM files are now only supported for reasons of compatibility. The statements listed in section “File processing” on page 190 should be used in their place.

@**EL**IM	Deletes records in an ISAM file. If all the records are deleted then - unlike @UNSAVE – the file name remains present in the catalog.	F mode L mode
@**F**ILE	Supplies a file name as the default value for @GET, @READ, @WRITE (Format 2), @SAVE, @OPEN (Format 2) and @ELIM. It is also possible to predefine a file name that only applies to the current work file (explicit local @FILE entry), or a file name which applies to all the work files (global @FILE entry).	F mode L mode
@**G**ET	Fully or partially reads an ISAM file from disk or tape into the current work file.	F mode L mode
@**OP**EN (Format 2)	Opens an ISAM file for processing directly on the disk. This file may already exist, may be created before being opened or be created as a copy of an existing SAM or ISAM file. It is only possible to open ISAM files for real processing in work file 0. This be empty or contain a file opened for real processing using @OPEN (format 2).	F mode L mode
@**REA**D	Fully or partially reads a SAM file from disk or tape into the current work file.	F mode L mode
@**SA**VE	Fully or partially writes the content of the current work file as an ISAM file to disk.	F mode L mode
@**U**NSAVE	Deletes a BS2000 file and the associated catalog entry.	F mode L mode
@**W**RITE (Format 2)	Fully or partially writes the content of the current work file as a SAM file to disk or tape.	F mode L mode

8.4 Old statements for processing POSIX files

These statements for processing POSIX files are now only supported for reasons of compatibility. The statements listed in section
“File processing” on page 190 should be used in their place.

@**XC**OPY	Reads a POSIX file which is stored in a POSIX file system into the current work file.	F mode L mode
@**X**OPEN	Opens a POSIX file which is stored in a POSIX file system and reads it into the current work file.	F mode L mode
@**XW**RITE	Writes the content of the current work file into a POSIX file. The work file is retained.	F mode L mode

8.5 Moving or positioning the work file

The following statements, which are primarily used in F mode, make it possible to move the required section of a work file into the screen for processing.

+	Moves forwards in the work file (toward the end of the file). The position can be moved forwards by a given number of lines or to a record with a specified record mark.	F mode
-	Moves backwards in the work file (toward the beginning of the file). The position can be moved backwards by a given number of lines or to a record with a specified record mark.	F mode
++	Moves to the end of the work file or to the last record with a specified record mark.	F mode
- -	Moves to the beginning of the work file or to the first record with a specified record mark.	F mode
<	Moves horizontally to the left in the work file, i.e. the data window can be moved through column-by-column to the left (toward the start of the record).	F mode
>	Moves horizontally to the right in the work file, i.e. the data window can be moved through column-by-column to the right (toward the end of the record).	F mode
<<	Moves horizontally to the start of the record in the work file, i.e. the data window is moved through column-by-column to the start of the record.	F mode

#	see @SETF statement	F mode
@**EN**D	In L mode, causes the current work file to be exited. Processing returns to the work file in which the @PROC statement activating the current work file was issued. In F mode, @END terminates the EDT session.	F mode L mode
@**O**N (Format 3)	Causes all records in which a hit is identified to be flagged with the specified record mark. In F mode, the work window is positioned at the first hit record.	F mode L mode
@**PRO**C (Format 1)	In L mode, switches to another work file. This work file then becomes the current work file.	L mode
@**SETF**	Simultaneously sets the vertical and horizontal position of the work window for a work file either with or without changing the current work file. In F mode, this statement may be abbreviated to # (if specified with operands).	F mode L mode
0..22	Switches to another work file.	F mode
$0..$22	Switches to another work file.	F mode

8.6 Treatment of line numbers

The following statements make it possible to adapt the line numbers and increments in a work file to meet current requirements. It is also possible to store line numbers in variables or number records sequentially.

@	See @SET (format 6).	F mode L mode
@+	The current line number is increased by the current increment or, in SEQUENTIAL mode (see the @EDIT statement), processing switches to the next current line.	L mode
@-	The current line number is reduced by the current increment or, in SEQUENTIAL mode (see the @EDIT statement), processing switches to the preceding line number.	L mode
@PAR	Defines the current increment by means of the INCREMENT operand.	F mode L mode
@**R**ENUMBER	The lines present in the work file are renumbered. It is possible to specify both the line number which is to accommodate the first line in the work file and the increment which is to be used for renumbering.	F mode L mode

@**SEQ**UENCE (Format 1)	Causes EDT to write a number in each line of a contiguous line range. A predefined number consisting of a maximum of 8 digits (possibly with leading zeros) is written to the first line of the line range. This also defines the number of digits in all the following numbers. All the following numbers are given by the total of the preceding number plus the predefined increment.	F mode L mode
@**SEQ**UENCE (Format 2)	Causes EDT to write the associated line number in each line of a contiguous line range. The line number is written as an 8-digit number without a decimal point.	F mode L mode
@**SEQ**UENCE (Format 3)	Causes EDT to examine the content of a column or contiguous range of columns in each line of a contiguous line range. It interprets the string it finds there as a binary number and checks whether the binary numbers form an ascending sequence.	F mode L mode
@SET (Format 3)	Assigns a value to a line number variable. This value may consist of: a line number specification, the value of an integer variable, the specification of a line number as a string or the binary value of the first 4 bytes in a string.	F mode L mode
@SET (Format 6)	Defines the current line number and the current increment or restores earlier values for the line number and increment.	F mode L mode

8.7 Creating, inserting and modifying texts

The following statements are used if it is necessary to make similar changes to text occupying a large range. They can be used in EDT procedures to automate frequently recurring changes.

@**CO**LUMN	Inserts or replaces text in existing work file lines or string variables as of the specified column position. Blanks at the end of a line are also deleted.	F mode L mode
@**CONV**ERT	In line ranges, converts lowercase characters to uppercase or uppercase to lowercase.	F mode L mode
@**CR**EATE (Format 1)	Creates a line with the specified content.	F mode L mode
@**CR**EATE (Format 2)	Assigns a string to a string variable.	F mode L mode

@**CR**EATE (Format 3)	Reads a string from the terminal or from SYSDTA and creates a line with its content.	F mode L mode
@**CR**EATE (Format 4)	Reads a string from the terminal or from SYSDTA and creates a string variable with its content.	F mode L mode
@**O**N (Format 6)	Searches for a string and replaces the hit string with the specified text.	F mode L mode
@**O**N (Format 7)	Searches for a string and inserts text before or after the hit string or replaces this.	F mode L mode
@**PRE**FIX	Prefixes each line or string variable in the specified range with a string.	F mode L mode
@**SDF**TEST	Checks whether a line range contains syntactically correct SDF commands or syntactically correct SDF statements.	F mode L mode
@**SEP**ARATE	Breaks a line or line range into multiple lines. The point at which the break takes place can be specified by a separator character or by a column position.	F mode L mode
@**SORT**	Sorts contiguous line ranges in the current work file in ascending or descending order. By specifying a column range, it is possible to restrict the sort operation to the relevant section of the record.	F mode L mode
@**SU**FFIX	Inserts a string at the end of each line or string variable in the specified range.	F mode L mode
@**T**ABS (Format 3)	Expands software tabulators in work files and string variables if a tabulator character and a corresponding tabulator position have been defined. (see @TABS, format 2).	F mode L mode
@**U**PDATE	This statement is now only supported in compatibility mode.	L mode

8.8 Copying and transferring lines

The following statements are used to copy or move larger text areas in cases where the intuitive method using statement codes in F mode would be too fiddly.

@**C**OPY (Format 2)	Copies lines from the current or another work file or string variable into the current work file.	F mode L mode
@**M**OVE	Transfers lines from the current or another work file or string variable into the current work file and deletes it at the original positions.	F mode L mode
@**O**N (Format 4)	Copies all the records marked with the specified record mark in the searched line ranges into the specified work file.	F mode L mode
@**O**N (Format 5)	Searches for a string and copies the hit lines into the specified work file..	F mode L mode

8.9 Deleting work files, lines, texts and record marks

The following statements provide various ways of deleting records, parts of records or entire work files.

@**D**ELETE (Format 1)	Fully or partially deletes lines and string variables.	F mode L mode
@**D**ELETE (Format 2)	Completely deletes work files.	F mode L mode
@**D**ELETE (Format 4)	Deletes record marks.	F mode L mode
@**DR**OP	Completely deletes the specified work files.	L mode
@**O**N (Format 8)	Deletes the hit string in the searched range.	F mode L mode
@**O**N (Format 9)	Searches for a string and deletes the content of a work file line or string variable before or after the hit string.	F mode L mode
@**O**N (Format 10)	Searches for a string and deletes the work file lines or the content of string variables which contain the search term.	F mode L mode

8.10 Comparing work files

The two statements for comparing work files differ primarily in the layout of the output and in the capability to compare only sections of work files.

@COMPARE (Format 1)	Compares two work files with one another either in full or in part. The results of the comparison can be sent to a work file, SYSOUT or SYSLST as required.	F mode L mode
@COMPARE (Format 2)	Compares the contents of two work files line-by-line. EDT stores the results in a work file. It is also possible to send the results to SYSLST and, in L mode, SYSOUT.	F mode L mode

8.11 Switching the work mode or operating mode

The following statements make it possible to switch between L mode and F mode or between Unicode mode and compatibility mode.

@DIALOG	In interactive mode, switches to the screen dialog.	F mode L mode
@EDIT (Format 1)	In interactive mode, switches from L mode to F mode.	F mode L mode
@EDIT (Format 2)	In the interactive mode's L mode, switches the input stream to terminal input. WRTRD is used for reading and the current line number is output as the prompt. If the statement is entered in F mode, operation first switches to L mode.	F mode L mode
@EDIT (Format 3)	In the interactive mode's L mode, this switches the input stream to input from SYSDTA. Reading is performed with RDATA. If the statement is entered in F mode, operation first switches to L mode.	F mode L mode
@MODE	Switches between Unicode mode and compatibility mode.	F mode L mode

8.12 Output lines and information

The following statements are used to output data and information. In most cases, it is possible to decide whether the output is to be written to a work file or to SYSOUT or SYSLST.

[n] **#**	Variant of the # statement (see the description of this statement). The nth last statement already executed is output in the statement line again.	F mode
#	Outputs the last statement already executed by EDT in the statement line again.	F mode
@FSTAT	Outputs a list of files in the BS2000 catalog to a work file or to SYSOUT or SYSLST as required.	F mode L mode
@LIMITS	Outputs the lowest and the highest assigned line numbers as well as the number of lines for the current work file.	F mode L mode
@LIST	Outputs ranges of a work file or string variables to SYSLST or at the printer.	F mode L mode
@LOG	Activates or deactivates logging of input in batch mode and interactive mode and controls the scope of logging.	F mode L mode
@ON (Format 1)	Searches for a string and outputs the content of every line or string variable in which a hit is found. In interactive mode, the output is written to SYSOUT and in batch mode it is written to SYSLST.	F mode L mode
@ON (Format 2)	Searches for a string and outputs the line numbers or the names of the string variables as well as the numbers of the columns in which the hit strings start.	F mode L mode
@PAGE	Generates a form feed at SYSLST.	F mode L mode
@PRINT	Outputs the content of the specified line ranges or string variables. In interactive mode, the output is written to SYSOUT and in batch mode it is written to SYSLST.	F mode L mode
@PROC (Format 2)	Outputs the number of the current work file, the numbers of all the free work files and the numbers of all the occupied work files.	L mode
@SHOW (Format 1)	Outputs a library's directory or a list of files from the BS2000 catalog or from a POSIX directory.	F mode L mode
@SHOW (Format 2)	Outputs a list of the character sets supported by XHCS. In interactive mode, it also indicates the character sets supported by the terminal.	F mode L mode

@**ST**ATUS	Outputs the EDT and system environment parameter settings as well as the values of line number and integer variables.	F mode L mode
@**SHIH**	Outputs the EDT statement buffer.	F mode L mode
@**TM**ODE	Outputs information about the task under which EDT is running. The information is output as a message.	F mode L mode

8.13 Interrupting or terminating EDT

The following statements interrupt or terminate EDT or execute system commands without exiting EDT.

@**EN**D	In L mode, causes the current work file to be exited. Processing returns to the work file in which the @PROC statement activating the current work file was issued. In F mode, @END terminates the EDT session or terminates the screen dialog after @DIALOG.	F mode L mode
@**EX**EC	Terminates the EDT session and loads and starts the specified program.	F mode L mode
@**H**ALT	Terminates the EDT session, the screen dialog after @DIALOG or EDT as a subroutine with or without transferring a text to the calling program.	F mode L mode
@**LO**AD	Terminates the EDT session and loads the specified program.	F mode L mode
@**RET**URN	In EDT procedures, terminates the execution of the procedure and returns to the point at which it was called. If the @RETURN statement is issued outside of an EDT procedure then the EDT session or, after @DIALOG, the screen dialog is terminated.	F mode L mode
@**SY**STEM	Interrupts (like K2) the EDT session or executes an operating system command without interrupting the EDT session.	F mode L mode

8.14 Runtime control in EDT procedures

The following statements are used in EDT procedures to control execution and to program loops and branches.

@**CON**TINUE	Does not perform any action. The statement is used to generate a line in EDT procedures which can be branched to with @GOTO.	L mode
@**GO**TO	In a @DO procedure, causes an unconditional jump to the specified line.	@PROC
@IF (Format 1)	In EDT procedures and in L mode checks whether EDT or DMS errors have occurred. Depending on the result, a specified string either is or is not processed as input.	L mode
@IF (Format 2)	In EDT procedures, compares strings, line numbers or integer variables with one another. Depending on the result, a specified string either is or is not processed as input.	L mode
@IF (Format 3)	In EDT procedures, checks whether EDT identified a hit the last time @ON was executed or whether the current work file is empty. Depending on the result, a specified string either is or is not processed as input.	L mode
@IF (Format 4)	In EDT procedures, checks which job and/or user switches are active and inactive. Depending on the result, a specified string either is or is not processed as input.	L mode
@IF (Format 5)	In EDT procedures or in L mode, identifies the currently set operating mode. Depending on the result, a specified string either is or is not processed as input.	L mode
@**RES**ET	Resets EDT and DMS error switches.	F mode L mode

8.15 Administering and executing EDT procedures

The following statements are used to start EDT procedures and supply parameters to them. Further statements are used to switch between different procedure levels and to initialize and modify variables.

@DO (Format 1)	Starts a @DO procedure, i.e. the text lines and EDT statements in the specified work file are processed.	F mode L mode
@DO (Format 2)	Activates and deactivates the logging of the read statements (see the PRINT operand in format 1 of the @DO statement) at any location within the procedure.	@PROC
@END	In L mode, causes the current work file to be exited. Processing returns to the work file in which the @PROC statement activating the current work file was issued. In F mode, @END terminates the EDT session or terminates the screen dialog after @DIALOG.	F mode L mode
@INPUT (Format 1)	Starts an @INPUT procedure from any file. The statements and/or records in the file are processed sequentially.	F mode L mode
@INPUT (Format 2)	Starts an @INPUT procedure from a SAM or ISAM file. The statements and/or records in the file are processed sequentially. This format is now only supported for reasons of compatibility and should no longer be used.	F mode L mode
@NOTE	Does not perform any action. The statement is used to insert comments in EDT procedures.	L mode
@PARAMS	Defines symbolic parameters which can be used in a @DO procedure.	@PROC
@PROC (Format 1)	Switches to another work file. This work file then becomes the current work file.	L mode
@SET (Format 1)	Assigns a value to an integer variable.	F mode L mode
@SET (Format 2)	Assigns a value to a string variable.	F mode L mode
@SET (Format 4)	Inserts the contents of an integer variable, the name of a string variable or the contents of a line number variable as of a given column in printable form in a work file line or string variable.	F mode L mode
@SET (Format 5)	Stores the date and time as of the desired column in a string variable or a work file line.	F mode L mode

8.16 Calling a user program

The following statements are used to load and start routines written by users or third-party suppliers in order to extend the EDT functionality.

@RUN	Calls a user routine. This statement is different from the statement of the same name in compatibility mode	F mode L mode
@UNLOAD	Unloads modules that have been loaded with @USE.	F mode L mode
@USE	Defines user statements by specifying a user statement symbol and the associated statement routine.	F mode L mode

8.17 Working with job variables

The following statements are used to process job variables.

@ERAJV	Deletes job variable entries from the catalog.	F mode L mode
@GETJV	Outputs the value of a job variable on the screen, writes it to a work file or assigns it to a string variable.	F mode L mode
@SETJV	Enters a job variable in the catalog or assigns it a value.	F mode L mode
@STAJV	Outputs the properties of job variables on the screen or to a work file.	F mode L mode

8.18 Working with S variables

The following statements are used to process S variables.

@GETLIST	Writes elements of an S list variable to the current work file.	F mode L mode
@GETVAR	Outputs the value of an S variable on the screen, writes it to a work file or assigns it to a string variable.	F mode L mode
@SETLIST	Assigns elements to an S list variable. In this case, lines are taken over from the current work file or from string variables.	F mode L mode
@SETVAR	Declares an S variable or assigns a value to an S variable.	F mode L mode

9 EDT statements (alphabetical)

The following sections contain the detailed descriptions of all the EDT statements listed in alphabetical order.

In the case of statements which contain special characters, the sequence followed is that defined in the character set EBCDIC.DF.04. The statement symbol @ in the statement name is ignored.

9.1 @< – Move data window to the left

The < statement moves the work file horizontally to the left, i.e. the data window can be moved through column-by-column to the left (toward the start of the record).

The column number as of which the records are displayed in the data window is output in the status display in the work window.

Operation	Operands	F mode
@<	[n]	

n Number of columns that the work window is to be shifted to the left. Values between `0` and `32768` are permitted for n. If the value specified for `n` is greater than the current column position then the position is set to the first column.

If `n` is omitted, EDT moves to the left by the current line length of the data window (depending on the employed terminal, the settings made for it in the @VDT statement and the visibility of the line number display).

If EDT is in `EDIT-LONG` mode then the statement is accepted and processed (as can be seen from the change in the status display). However, the change is not visible until `EDIT-LONG` mode is exited.

Example

The data window starts at column 10 (see status display).

```
   1.00 ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····························
   2.00 DONALD    WALTSTREET 8   DISNEYLAND<·····································
   3.00 GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····························
   4.00 LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····························
   5.00 MANUELA   POSTWEG 3      80123 MUENCHEN<·····························
   6.00 ···········································································
<9·····························································0001.00:00010(00)
```

The data window is to be shifted 9 columns to the left.

```
   1.00 BERGER   ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····················
   2.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   3.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   4.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···········································································
·····························································0001.00:00001(00)
```

This operation causes the data window to start at column 1.

9.2 @<< – Move data window to the start of the record

The << statement moves the work file horizontally to the start of the record, i.e. the data window can be moved through column-by-column to the left as far as the start of the record. After this operation, the displayed section of the records starts at column 1. The column number is output in the work window's status display.

Operation	Operands	F mode
@<<		

If EDT is in EDIT-LONG mode then the statement is accepted and processed (as can be seen from the change in the status display). However, the change is not visible until EDIT-LONG mode is exited.

Example

The data window starts at column 10 (see status display).

```
   1.00 ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····························
   2.00 DONALD    WALTSTREET 8   DISNEYLAND<·····································
   3.00 GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····························
   4.00 LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····························
   5.00 MANUELA   POSTWEG 3      80123 MUENCHEN<·····························
   6.00 ···········································································



<<·····························································0001.00:00010(00)
```

The data window is shifted to the left as far as column 1.

```
   1.00 BERGER   ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····················
   2.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   3.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   4.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···········································································




···············································································0001.00:00001(00)
```

This operation causes the data window to start at column 1.

9.3 @+ – Increase the current line number

The @+ statement increases the current line number by the current increment or, in SEQUENTIAL mode (see the @EDIT statement), processing switches to the next current line.

Operation	Operands	L mode
@+	[:[text]]	

text

EDT statement or data input which is executed or inserted in the new current line after the current line number has been increased. The string is treated as if it had been entered at the prompt in L mode. In particular, the decision to interpret the text as data input or as a statement is made in accordance with the same rules (for more information, see section "L mode" on page 126).

The text operand starts immediately after the character ':', i.e. any specified blanks form part of the operand and are taken over into the line in the case of data input.

If text is not specified (although the colon is), then an empty line (line of length 0) is inserted.

If no operand is specified then only the current line is modified.

The indirect specification of operands is not permitted for this statement.

9.4 + – Move data window forwards

The + statement moves forwards in the work file (toward the end of the file). The position can be moved forwards by a given number of characters or to a record with a specified record mark.

Operation	Operands	F mode
+ DUE		
+ F3		
+	{ n | ([m[,...]]) }	

If the statement is sent without operands by means of the DUE key or a function key other than F3 then the position moves forwards by the number of records visible in the data window. Any column counters or lines hidden by messages, for example, are taken into account.

If the statement is sent without operands by means of F3 then the position moves to the next record having any record mark (1..9). The statement +F3 is therefore equivalent to +() DUE (see below).

n — Number of lines to be scrolled through forwards. Values between 0 and 99999999 are permitted for n. However, forwards scrolling stops when the last record in the work file is visible in the first screen line.

The value of n determines the number of records by which the data window can be moved forwards independently of the currently set increment in the line number display or gaps in the record numbering.

m — Is one of the possible record marks (1..9) to which the position may be moved. It is possible to specify multiple record marks which must be separated by commas. The position moves forwards to the next record with one of the specified record marks - this is displayed in the first screen line of the data window. Marks with special functions (see section “Record marks” on page 45) are ignored here.

If m is not specified then the position moves to the next record having any record mark (1..9).

Note

If the statement is sent with F3 then it is important to make sure that only statement codes that can be sent with F3 are specified simultaneously (see section "Statement codes in F mode" on page 109). Otherwise, the operation is aborted during the analysis of the statement codes and the + statement is not executed.

Example

A column counter is displayed in the top data window which has been reduced with @SPLIT. The last line in the data window is hidden by the message EDT0901.

```
        ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   1.00 BERGER   ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····················
   2.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   3.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
%  EDT0901 NO MATCH IN RANGE
+·····························································0001.00:00001(00)
   1.00 ·····································································
 
 
 
·····························································0000.00:00001(04)
```

The + statement is to be used to scroll forwards.

```
        ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   4.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ·····································································
   7.00 ·····································································
·····························································0004.00:00001(00)
   1.00 ·····································································
 
 
 
·····························································0000.00:00001(04)
```

The position scrolls forwards to the first line that was not previously visible (4.00).

For an example of moving to a record with a record mark, see the description of the @ON statement, format 4.

9.5 ++ – Move to the last (marked) record in the work file

The ++ statement moves to the end of the work file or to the last record with a specified record mark.

Operation	Operands	F mode
++ DUE		
++ F3		
++	([m[,...]])	

If the statement is sent without operands by means of the DUE key or a function key other than F3 then the work file is positioned in such a way that the last record of the work file is displayed in the last screen line of the data window (in contrast, +99999999 causes the last record in the work file to be displayed in the first screen line of the data window).

If the statement is sent without operands by means of F3 then the position moves to the last record having any record mark (1..9). The statement +F3 is therefore equivalent to ++() DUE.

m
: Is one of the possible record marks (1..9) to which the position may be moved. It is possible to specify multiple record marks which must be separated by commas. The position moves forwards to the last record with one of the specified record marks - this is displayed in the first screen line of the data window. Marks with special functions (see section “Record marks” on page 45) are ignored here.

  If m is not specified then the position moves to the last record having any record mark (1..9).

Note

If the statement is sent with F3 then it is important to make sure that only statement codes that can be sent with F3 are specified simultaneously (see section on statement codes). Otherwise, the operation is aborted during the analysis of the statement codes and the ++ statement is not executed.

9.6 $0..$22 – Change work file

This statement causes EDT to switch to another work file.

Operation	Operands	F mode
$0..$22		

EDT displays the work file selected with the $0..$22 statement in the work window in which the statement was entered. The line position and column position are set to the values that were previously valid in the newly set work file. If the work file has not yet been used then the default values apply.

The @SETF statement can also be used to change work file and, at the same time, set the position to any required line and column.

Example

```
   1.00 This statement switches EDT to another<................................
   2.00 work file<.............................................................
   3.00 .......................................................................



$7 ...........................................................0001.00:00001(00)
```

Statement $7 is entered in order to switch to work file 7.

```
.............................................................0000.00:00001(07)
```

Work file 7 is displayed in the work window (see status display).

9.7 @– – Decrease the current line number

The @- statement reduces the current line number by the current increment or, in SEQUENTIAL mode (see the @EDIT statement), processing switches to the preceding current line.

Operation	Operands	L mode
@–	[:[text]]	

text

EDT statement or data input which is executed or inserted in the new current line after the current line number has been decreased. The string is treated as if it had been entered at the prompt in L mode. In particular, the decision to interpret the text as data input or as a statement is made in accordance with the same rules (for more information, see section “L mode” on page 126).

The text operand starts immediately after the character ':', i.e. any specified blanks form part of the operand and are taken over into the line in the case of data input.

If text is not specified (although the colon is), then an empty line (line of length 0) is inserted.

If no operand is specified then only the current line is modified.

The indirect specification of operands is not permitted for this statement.

9.8 – – Move data window backwards

The - statement moves backwards in the work file (toward the start of the file). The position can be moved backwards by a given number of characters or to a record with a specified record mark.

Operation	Operands	F mode
– DUE		
– F3		
–	{ n / ([m[,...]]) }	

If the statement is sent without operands by means of the DUE key or a function key other than F3 then the position moves backwards by the number of records visible in the data window. Any column counters or lines hidden by messages, for example, are taken into account.

If the statement is sent without operands by means of F3 then the position moves backwards to the next record having any record mark (1..9). The statement -F3 is therefore equivalent to -() DUE (see below).

n — Number of lines to be scrolled through backwards. Values between 0 and 99999999 are permitted for n. However, backwards scrolling stops when the first record in the work file is visible in the first screen line.

The value of n determines the number of records by which the data window can be moved backwards independently of the currently set increment in the line number display or gaps in the record numbering.

m — Is one of the possible record marks (1..9) to which the position may be moved. It is possible to specify multiple record marks which must be separated by commas. The position moves backwards to the next record with one of the specified record marks - this is displayed in the first screen line of the data window. Marks with special functions (see section “Record marks” on page 45) are ignored here.

If m is not specified then the position moves to the next record having any record mark (1..9).

Note

If the statement is sent with F3 then it is important to make sure that only statement codes that can be sent with F3 are specified simultaneously (see section on statement codes). Otherwise, the operation is aborted during the analysis of the statement codes and the - statement is not executed.

Example

The increment 0.1 is set in the data window.

```
      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
  10.00 BERGER   ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····················
  10.10 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
  10.20 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
  10.30 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
  10.40 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
  10.50 ·····························································
  10.60 ·····························································
  10.70 ·····························································
-3·····························································0010.00:00001(01)
```

Backwards scrolling is to be performed using -3.

```
      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   9.70 ÅNGSTRØM ANDERS    STERNWARTE     STOCKHOLM<··························
   9.80 BASLER   MARIO     SÄNBENER STR.1 80321 MUENCHEN<·····················
   9.90 BAYER    ALOIS     OTTOSTR.4      80123 MUENCHEN<·····················
  10.00 BERGER   ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····················
  10.10 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
  10.20 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
  10.30 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
  10.40 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
  10.50 ·····························································
  10.60 ·····························································
·····························································0009.70:00001(01)
```

The position moves backwards 3 records (to line 9.70).

9.9 – – – Move to the first (marked) record in the work file

The -- statement moves to the start of the work file or to the first record with a specified record mark.

Operation	Operands	F mode
– –DUE		
– –F3		
– –	([m[,...]])	

If the statement is sent without operands by means of the DUE key or a function key other than F3 then the position moves to the first record in the work file, i.e. the first record in the work file is displayed in the first line in the data window.

If the statement is sent without operands by means of F3 then the position moves to the first record having any record mark (1..9). The statement
– –F3 is therefore equivalent to – –()DUE.

m
: Is one of the possible record marks (1..9) to which the position may be moved. It is possible to specify multiple record marks which must be separated by commas. The position moves backwards to the first record with one of the specified record marks - this is displayed in the first screen line of the data window. Marks with special functions (see section “Record marks” on page 45) are ignored here.

  If m is not specified then the position moves to the first record having any record mark (1..9).

Note

If the statement is sent with F3 then it is important to make sure that only statement codes that can be sent with F3 are specified simultaneously (see section on statement codes). Otherwise, the operation is aborted during the analysis of the statement codes and the -- statement is not executed.

9.10 @> – Move data window to the right

The > statement moves the position horizontally in the work file, i.e. the data window can be moved through column-by-column to the right (toward the end of the record and beyond).

The column number as of which the records are displayed in the data window is output in the status display in the work window.

Operation	Operands	F mode
@>	[n]	

n Number of columns that the work window is to be shifted to the right. Values between `0` and `32768` are permitted for `n`. The position can be moved so far to the right that the last column of the screen line displays the maximum column position which EDT permits for a record (32768). This applies independently of whether the work file actually contains any records of this length.

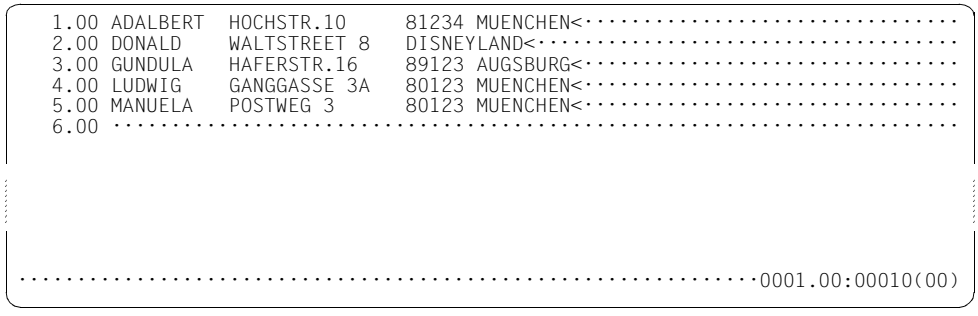
If `n` is omitted, EDT moves to the right by the current line length of the data window (depending on the employed terminal, the settings made for it in the @VDT statement and the visibility of the line number display).

If EDT is in `EDIT-LONG` mode then the statement is accepted and processed (as can be seen from the change in the status display). However, the change is not visible until `EDIT-LONG` mode is exited.

Example

```
   1.00 BERGER   ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····················
   2.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   3.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   4.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································································
>9····· ·····················································0001.00:00001(00)
```

The data window is shifted 9 columns to the right.



This operation causes the data window to start at column 10.

9.11 @: – Declaring a statement symbol

The @: statement is used to define a new statement symbol.

Operation	Operands	F mode, L mode
@:	spec	

spec Special character for the new statement symbol.

If the spec operand is not a valid special character then @: is rejected with the error message EDT3952.

The current range symbol (see @RANGE) may not be used for the spec operand and is rejected with the error message EDT4315.

If this statement is issued in F mode then it also must be preceded by the previously valid statement symbol.

When EDT starts, @ is the current statement symbol.

Caution

If the spec operand is assigned one of the special characters <, > (only in F mode), +, –, $, %, * or ? (in F and L mode) then the statements may in some cases be ambiguous and undesired program behavior may occur.

If the special character : is used for spec then it is no longer possible to undo the setting since, from this moment onwards, a sequence of colons at the start of a line is interpreted as a sequence of statement symbols.

Example

```
3.     @print ------------------------------------------------------------ (1)
1.0000 This statement allows the user to declare a new
2.0000 statement symbol.
3.     @:! ------------------------------------------------------------- (2)
3.     @print ------------------------------------------------------------ (3)
4.     !print ------------------------------------------------------------ (4)
1.0000 This statement allows the user to declare a new
2.0000 statement symbol.
3.0000 @print
4.     !:@ ------------------------------------------------------------- (5)
4.
```

(1) @PRINT is used to output the content of the work file.

(2) ! is declared as the new statement symbol.

(3) @PRINT is now not interpreted as a statement but as text.

(4) !PRINT outputs the content of the work file.

(5) @ is declared as the statement symbol again.

9.12 # – Output the last statement

The # statement can be used to output one of the last statements already executed by EDT in the statement line again. This does not apply to any of the scrolling statements or the statements used to change work file.

Operation	Operands	F mode
[n] #		

n n specifies the depth, i.e. how far back in the processing sequence the statement to be output lies. Values between 1 and 2048 are permitted for n.

The # or 1# statement outputs the last statement already executed by EDT in the statement line again. 2# outputs the second last statement. If the # statement is entered more than once successively then the pointer in the statement buffer is preset to the specified position each time. If the start of the buffer has been reached then the statement line remains empty. If a # statement then follows, the position returns to the last saved statement (end of the buffer). After each entry other than # or after an empty entry, the end of the buffer is taken as the starting point again.

If the statement to be displayed is longer than the command line then up to 3 command lines can be displayed. If even then it is not possible to display the statement in full, it is truncated and no message is issued.

If a communications character set other than the current one was defined when a statement was entered then the statement is converted accordingly for output in the statement line. If, in such a case, it is not possible to convert individual characters then the message EDT5453 is issued. The statement is output nevertheless and the non-converted characters are replaced by question marks '?'.

The statement buffer can accommodate a maximum of 2048 statements independently of their various lengths.

At least one character in the statement line must be overwritten, modified or added if the content of the line is to be sent as a statement.

This operation takes no account of whether a statement was entered in the upper or lower part of a split screen. Statements are stored in the statement buffer independently of the work file to which the statement was applied. Statements in a statement sequence (statements separated by ';') are stored individually.

If a # statement is entered in a statement sequence then after # has been executed, processing is aborted and the last executed statement is output. Any statements located after # in the statement sequence are not executed.

Note

In F mode, the statement # followed by operands is an abbreviation of the @SETF statement and is used to position the data window. This statement should not be confused with the one described here.

The statement @SHIH (Show Input History) can be used to output the entire EDT statement buffer to a work file. It is then possible to use statement code K to copy individual statements into the statement line.

9.13 @AUTOSAVE – Automatic saving

The @AUTOSAVE statement activates the automatic time-controlled saving of work files.

Operation	Operands	F mode / L mode
@AUTOSAVE	{[**ID**=name] [[,] **T**IME=n] [**ON**] / **O**FF}	

name — Freely selectable identifier for the autosave files which is included in the names assigned to the autosave files (default value: `EDT`).

n — Time interval in minutes(`0..255`) between a manual or automatic save and the next automatic save (default value: `5`).

If `TIME=0` then a save is performed after every dialog step.

ON — Automatic saving is activated (default value).

OFF — Automatic saving is deactivated. All the autosave files are deleted.

The @AUTOSAVE statement is only effective in interactive mode, i.e. data is only saved in interactive mode. The statement is ignored in batch mode and no message is issued.

When an EDT session starts, the autosave function is deactivated.

Under certain circumstances, it may not be possible to save very long records (close to `32768`). In such cases, the maximum possible length of the record in question is saved and the warning `EDT2405` is issued.

A save operation is performed each time the autosave function is activated or after every dialog step, i.e. before the next prompt, if the autosave function is activated and the time defined when it was activated has elapsed since the last automatic save. All non-empty work files whose content is not already present as a disk file (either explicitly created by the user or created by the autosave function in the form of an autosave file) are saved. ISAM files opened for real processing are not saved.

The names of the autosave files are formed as follows:

`S.name.yyyy-mm-dd.hhmmss.SAVEnn`

The specification `yyyy-mm-dd.hhmmss` is the point in time at which the autosave function was activated. The specification nn is the number of the current work file.

An associated autosave file is deleted, if:

– the work file no longer contains any records, (e.g. @DELETE)

– the contents of the file have been explicitly saved as a file or library element. The possible statements are: @WRITE, @SAVE, @XWRITE and @CLOSE.

All the autosave files are deleted when the statement @AUTOSAVE OFF is issued, when the EDT session is terminated with one of the statements @HALT, @END, @EXEC or @LOAD or when control returns to the main program.

The save files are not deleted on abnormal termination or if EDT is exited with K2 or the @SYSTEM statement without a return.

Note

The content and character set of an individual work file can be restored by issuing the following statement in an empty work file.

```
@COPY FILE=S.name.yyyy-mm-dd.hhmmss.SAVEnn,KEY=LINENUMBER
```

9.14 @BLOCK – Set block mode

The @BLOCK statement activates or deactivates EDT's blocked input/output mode (BLOCK mode).

When BLOCK mode is active, it is possible, in L mode, to use a single entry at the terminal to create multiple lines or input multiple statements for sequential execution or enter a mixture of the two. The individual lines and/or statements must be separated using the LZE character.

Operation	Operands	F mode / L mode
@BLOCK **@BK**	[{ **ON** / **OF**F }]	

ON Activates BLOCK mode (default value).

OFF Deactivates BLOCK mode.

When an EDT session starts, BLOCK is activated by default.

In BLOCK mode, the maximum number of lines that can be entered in a block is the number that can be displayed on a screen page.

If the entered block contains an illegal statement then this is output together with the current line number and the corresponding error message at the time of its execution. The remainder of the block is then executed.

If an entered block contains a @BLOCK OFF statement then the remaining statements or data input in the block are ignored.

In batch mode, the @BLOCK statement is ignored.

When @DO procedures are called, BLOCK mode is set to OFF. If the @DO procedure is exited again, BLOCK mode is restored to the status it had before the @DO procedure was called.

9.15 @CHECK (format 1) – Check lines

This statement can be used to log every line that is created or modified in a work file or a string variable by a statement. In interactive mode, the line in question is output to SYSOUT and in batch mode it is output to SYSLST. The @CHECK statement can also be used to check the line length (number of characters per line) while taking account of tabulator expansion.

Operation	Operands	L mode
@CHECK	[{ **ON** / **OF**F }] [,] [col]	

ON — Activates CHECK mode (default value). If CHECK mode is activated then every line that is created or modified in a work file or a string variable by one of the following statements is written to SYSOUT: @COLUMN, @COPY (format 2), @CREATE, @MOVE, @ON (formats 7 to 10), @PREFIX, @SET (formats 2, 4 and 5), @SEPARATE, @SEQUENCE (formats 1 and 2), @SUFFIX.

OFF — Deactivates CHECK mode.

col — Specifies the number of characters per line for the check of the line length. In particular, it displays any cases in which the predefined line length is exceeded due to possible tabular expansion EDT checks the number of characters in every line which is newly entered or created by one of the following statements: @+, @–, @IF, @SET (format 6). The number of characters per line is checked independently of the current CHECK mode setting.

If a line is longer than the value specified in col then the line is nevertheless created and EDT outputs the message EDT2901 to indicate that the predefined number of characters per line has been exceeded. The default value of col corresponds to the maximum possible value of 32768 characters per line while the minimum possible value of col is 1 character per line.

The col value can also be modified using the @TABS statement.

When EDT starts, CHECK mode is deactivated.

The @CHECK statement is only effective in L mode. Any temporary switch to F mode deactivates the CHECK mode. This does not change the current value of col.

Specifying ON or OFF has no effect on the current value of col. Consequently, the current value of col is not modified if only ON or OFF is specified. To reset col to the default value 32768, it is necessary to specify this default value explicitly (@CHECK [ON,]32768 or @CHECK OFF,32768).

9.16 @CHECK (format 2) – Check lines for convertibility

This format of the @CHECK statement can be used to check whether the specified range in the current work file or range of string variables can be converted into the target character set without loss.

Operation	Operands	F mode, L mode
@**CH**ECK	[{ lines / svars } [,...] [:cols[:]]] [,] **C**ODE = { name / ***E**DT / ***F**ILE } [,**M**ARK[=m]] [,**L**ENGTH = { int / ***F**ILE }]	

lines — One or more line ranges to be checked. If the specified range contains no lines or only empty lines then the result of the check is positive (empty lines can always be written without loss).

svars — One or more ranges of string variables to be checked.

cols — Contiguous column range for checking in the current work file or in the specified string variables.

If the range specification contains only a single column specification, this indicates the range from the specified column through to the end of the line. If the first column specification is greater than the line length then the line or string variable is ignored.

If no column range is specified then the entire line or string variable is checked.

CODE= — Specifies, either directly or symbolically, the character set for which the check is to be performed.

name — Name of the character set for which the check is to be performed. The specified line range or range of string variables is converted into this character set for test purposes. Depending on the selected option, faulty lines are either marked or output to SYSOUT.

*EDT — The character set that is set for the current work file should be used for the check. Since a work file may not contain any records with characters which are invalid in the work file's character set, the specification of *EDT is only of any value if only a check for lines of excess length is to be performed (see the LENGTH operand).

*FILE The character set entered in the catalog for the open file or open library element or which was specified in the CODE operand in the @OPEN statement is to be used for the check. If no file or library element is open then the specification of *FILE is rejected with error message EDT5467. If the character set *NONE is entered for the file then the check is performed using the character set EDF03IRV.

MARK= The lines which cannot be converted without loss into the specified character set should be assigned a record mark. This option is only permitted if only line ranges in the current work file are to be checked. If MARK is not specified then the lines or string variables which cannot be converted without loss are output to SYSOUT.

m Record mark (1..9) used to mark the lines which cannot be converted without loss into the specified character set. If m is not specified, record mark 1 is used.

LENGTH= This specifies a maximum length (in bytes) which may not be exceeded when the lines or string variables are converted. Since, when conversion is performed into a Unicode character set, many characters are coded by multiple bytes, a line may become longer on conversion. Lines or string variables which exceed the specified maximum length on conversion are therefore considered to contain an error.

int Explicitly defines a maximum length (1..32768).

*FILE The maximum length is calculated from the catalog entry for the open file or open library element. A value is selected which guarantees that all the checked records can be copied without loss into the file when all the other relevant file properties (e.g. the file type and record format) are taken into account. If no file or library element is open then the value 32768 is used.

If neither lines nor svars is specified then the entire current work file is checked.
A line or string variable is considered to contain an error for the purposes of the check performed by @CHECK if, after conversion, it is either longer than the value specified in LENGTH or if it contains characters which would have to be mapped to any substitute character which may have been specified (see the statement @PAR SUBSTITUTION-CHARACTER) when converted into the specified character set.

If EDT does not identify any such defective lines or string variables or if they are only found to be of excess length (see below) then the user can be certain that a conversion into the specified character set is possible even if no substitute characters have been specified. Otherwise, the user can decide whether to specify a substitute character (if this has not already been done) and accept the resulting loss of information or to modify the relevant lines or string variables in order to permit loss-free conversion.

Lines or string variables which are found to be excessively long are truncated when written to a file, a job variable or an S variable if the checked length is of physical relevance (e.g. in the case of files with a fixed record length). In all cases, truncation is performed at a valid boundary between two characters. The length that is actually written may therefore be a maximum of 3 bytes shorter than the checked length. If truncation is not acceptable, the user must subdivide or shorten the identified lines in a meaningful way.

The function not only marks and/or outputs the lines or string variables in which errors are detected but also issues a message summarizing the result of the check. If only invalid characters are detected, the message `EDT5453` is output and if only length overruns are detected, the message `EDT5462` is output. If both invalid characters and length overruns occur, the message `EDT5456` is output.

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

9.17 @CLOSE – Write back and close a file

@CLOSE may be used to write back an open file, close this file and then delete the work file.

Operation	Operands	F mode, L mode
@CLOSE	[**N**OWRITE | **C**ODE = name | ***E**DT | ***F**ILE]	

NOWRITE The work file is deleted and is not written back. The opened file or library element is closed unchanged. If a file has been opened for real processing in work file 0 (see @OPEN, format 2) then `NOWRITE` has no effect.

CODE= The operand controls the character set in which the work file is to be written. In the case of a file opened for real processing in work file 0 the operand has no effect.

If the operand is not specified and if the character set of the SAM file, ISAM file or library element or the character set used when opening a POSIX file differs from that of the work file then, in batch mode, the message `EDT5457` is output, no write operation is performed and the file remains open. In interactive mode, the query

```
%  EDT0915 CONVERT TO FILE CCS (&00)? REPLY (Y=YES; N=NO)?
```

is output. If the user responds `Y` then a conversion to the file's character set is performed before the write operation. If the user responds `N` then the work file's character set is used.

- name — Character set that is to be used for writing. The name of a valid character set must be specified (see section “Character sets” on page 47).
- *FILE — Before the write operation, the work file is converted into the character set of the existing SAM file, ISAM file or library element or into the character set used when opening a POSIX file. If this character set was `*NONE` then `EDF03IRV` is used.
- *EDT — The work file's character set is used for writing irrespective of whether any file that may exist has a different character set.

The @CLOSE statement is rejected with error message `EDT5177` if the current work file does not contain a file or library element opened with @OPEN or @XOPEN.

After the close operation, the character set used for writing is entered in the catalog for SAM files, ISAM files and library elements.

If the work file is converted before writing and if it contains characters which are invalid in the character set used by the file that is to be written then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the file is not written, it remains open and error message EDT5453 is output. The user can then define a substitute character or modify the character set for writing and run @CLOSE again.

If the work file contains lines that are too long for the file that is to be written (e.g. if the file has a fixed record length) or if the conversion operation creates any such records (possible in the case of Unicode character sets), then the write operation is aborted with the message EDT5444.

When ISAM files are written, the ISAM key is formed from the line number if KEY=LINENUMBER or KEY=IGNORE was specified when the file was opened. If KEY=DATA was specified when the file was opened then the ISAM key is taken over from the data area. In this case, the user must make sure that the sequence of work file records corresponds to the sequence of ISAM keys as otherwise the write operation will be rejected with the message EDT4208 (DMS error code 0AAB).

The definition of any secondary keys in an ISAM file (in a secondary index) is retained after @CLOSE unless the key fields have been modified inconsistently in the data area. In this case, the message EDT5246 is output and the secondary index is deleted.

If, during the processing of an opened ISAM file, the character set is changed either from or to UTF16 or if this occurs implicitly due to a corresponding specification in the CODE operand then the file cannot be written back since this would modify the length of the key field. In this case, the @CLOSE statement is rejected with the error message EDT5468.

After a SAM or ISAM file has been closed with @CLOSE, EDT usually releases the file's no longer required disk storage space. However, this can be prevented by setting job switch 7.

If the current version number was specified when an ISAM file was opened for real processing (see @OPEN, format 2) whereas the AS operand was not specified then the current version number (which has been incremented by 1) is displayed after @CLOSE.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Note

When @CLOSE is run, an implicit @DELETE (format 2) is executed for the current work file. This resets a number of work file properties to their initial values.

Example

```
@OPEN FILE=FILE1 ------------------------------------------------------ (1)
<EDT statements> ------------------------------------------------------ (2)
@CLOSE CODE=UTFE ------------------------------------------------------ (3)
```

(1) The file `FILE1` is opened and read into the current work file in the file's character set.

(2) The current work file is processed.

(3) The file `FILE1` is opened and read into the current work file in the file's character set. In the BS2000 catalog, `FILE1` is assigned the attribute `CODED-CHARACTER-SET=UTFE`. The current work file is deleted.

9.18 @CODENAME (format 1) – Define the character set for work files and string variables

Format 1 of the @CODENAME statement can be used to define the character set used by EDT for a work file or a string variable. The definition made in @CODENAME takes priority over the implicit selection of a character set by EDT, for example on the basis of an entry in the file catalog.

Operation	Operands	F mode, L mode
@CODENAME	name [, { **L**OCAL / **G**LOBAL / $0...$22 / #S0...#S20 }] [, **F**ORCE = { **Y**ES / **N**O }]	

name — Name of the character set that is to be defined. The character set name must be known in XHCS; otherwise, the statement is rejected with message EDT4980. The specified character set is defined for one or more work files or for the string variable determined by means of the other operands.

LOCAL — The specified character set is defined for the current work file. If the work file is not empty and the operand FORCE=YES is not specified then the data it contains is converted into the new character set.

GLOBAL — The specified character set is defined globally for all EDT work files. All non-empty work files are converted unless the operand FORCE=YES has been specified. This setting does not apply to character sets for string variables.

The operand is primarily used if EDT is called as a subroutine via the old subroutine interface (V16 format). If EDT recognizes that the same character set is defined for all the work files then this character set is entered in the global control block EDTPARG at the old subroutine interface. If the calling program evaluates this entry, it is therefore advisable to use the GLOBAL operand to define the same character set for all the work files (and then not to change these again afterwards).

For the behavior of the old subroutine interface if no global character set is defined and further details concerning the use of character sets at the subroutine interface, see the Subroutine Interfaces User Guide [1].

$0..$22 — The specified character set is defined for the specified work file. If the work file is not empty and the operand FORCE=YES is not specified then the data it contains is converted into the new character set.

#S0..#S20 The specified character set is defined for the specified string variable. If the operand FORCE=YES is not specified then the data present in the string variable is converted into the new character set.

FORCE=

NO The data in the specified work file or string variable is converted into the specified character set.

YES The specification FORCE=YES is only permitted for 7 and 8-bit character sets and results in the relabeling of the specified work file or string variable, i.e. the content of the work file or string variable remains unchanged (as a byte sequence) but is reinterpreted in the specified character set. This function is used to correct errors in the assignment of character sets, for example due to incorrect entries in the DMS catalog. In the case of Unicode character sets, the statement is rejected with the message EDT5494.

If the work file is converted and if it contains characters which cannot be displayed in the target character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @CODENAME statement is rejected and error message EDT5453 is output. Specifying GLOBAL does not, in this case, convert back work files which have already been converted.

If a character set is assigned to a work file or a string variable using @CODENAME then this assignment applies until it is explicitly modified by another @CODENAME statement or until the work file or string variable has been completely deleted (e.g. with @DELETE, format 2). This means that all the data that is copied or read into the work file or character set is converted into this character set (see section “Character sets in work files” on page 54).

If an ISAM file is opened for real processing in work file0 by means of the @OPEN statement (format 2) then every @CODENAME statement which applies explicitly or implicitly for this work file is rejected with message EDT5452.

In the case of an opened, existing file for which the character set *NONE is entered, calling the @CODENAME statement always causes the character set used for writing to be entered explicitly in the catalog when the file is written back with @CLOSE or @WRITE.

9.19 @CODENAME (format 2) – Define the communications character set

Format 2 of the @CODENAME statement can be used to define the communications character set used by EDT. The definition made in @CODENAME takes priority over the implicit selection of a character set by EDT, for example on the basis of the employed terminal.

Operation	Operands	F mode, L mode
@CODENAME	[{ name / ***A**UTO } ,**TERM**INAL]	

name — Name of the character set that is to be defined. The character set must be known in XHCS. The character set must be supported by the employed terminal; otherwise, the statement is rejected with message EDT5487.

If a communications character set is specified explicitly then it cannot be selected automatically by EDT (see section “Communications character set” on page 53).

*AUTO — Activates automatic selection of the communications character set by EDT (see section “Communications character set” on page 53).

If no operand is specified then only the character set defined using /MODIFY-TERMINAL-OPTIONS is used as the communications character set. This is also the default setting when EDT is started.

The specified character set is used as the communications character set for data exchange with the terminal and the operand has no direct influence on the coding in the work files (see section “Character sets” on page 47). However, empty work files which have no character set and are only filled with data as a result of direct user input at the terminal are initially assigned this character set. More specifically, if the automatic selection of the communications character set is active and a modern Unicode-compatible emulation is being used then work files which are filled with data in this way are always assigned the character set UTFE. This can only be prevented by the earlier explicit assignment of a work file character set.

In batch mode, this statement is ignored.

9.20 @COLUMN – Insert text and delete blanks at end of line

The @COLUMN statement modifies the content of existing work file lines or string variables.

During the first stage, new text is inserted or existing text is overwritten as of the specified column. Then all the blanks are deleted (as far as the next character which is not a blank) from right to left starting at the last character in the work file line or string variable. A work file line which only consists of blanks remains present as an empty line in the work file. A string variable which only contains blanks becomes an empty string variable after the delete operation.

Operation	Operands	F mode, L mode
@COLUMN	col **ON** { lines / svars } [,...] [{ **C**HANGE / **I**NSERT }] [:] string	

col — Column as of which text is to be replaced or inserted.

lines — One or more line ranges in which text is to be inserted or replaced. Only existing lines are processed.

svars — One or more ranges of string variables in which text is to be inserted or replaced.

CHANGE — The string specified with `string` replaces the existing text as of column `col` (default value).

INSERT — The string specified with `string` is inserted as of column `col`.

: — This specification is obligatory if neither `CHANGE` nor `INSERT` is specified in order to clearly separate the range specification from the `string`.

string — String which is inserted or which replaces existing text as of the specified column in every line of a specified line range. It is also permissible to specify an empty string.

The string is converted into the character set used by the work file or string variable. If the string contains characters which cannot be displayed in the target character set then these are replaced by a substitute character if such a character has been specified.
(see @PAR SUBSTITUTION-CHARACTER); otherwise, the @COLUMN statement is rejected and the error message `EDT5453` is issued.

If the column as of which the text is to be inserted is located after the previous line end then the intervening columns are filled with blanks.

No text is replaced or inserted if this would cause the work file or string variable line length to exceed the permitted maximum of 32768 characters. Instead, in this case, the message EDT5474 is output. EDT does not check whether deleting blanks at the end of a line would restore the line to a permitted value.

If errors occur during processing (EDT5453 or EDT5474) then the statement is aborted. Any lines and/or string variables which have been successfully modified up to this point retain their changes.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Note

This statement can be used to delete blanks at ends of lines without knowing the line length. If an empty string is specified as the replacement text in the @COLUMN statement then no text is replaced in the lines. The right-to-left delete operation causes blanks to be deleted at the ends of the lines.

Example

```
   1.00 126790<·····················································
   2.00 348<·····················································
   3.00 ··························································
column 3 on 1:2 ·····························································0001.00:00001(01)
```

The content of column 2.00 is to be entered in line 1.00 as of column 3. The old content of line 1.00 is therefore overwritten.

```
   1.00 123480<·····················································
   2.00 348<·····················································
   3.00 ··························································
column 5 on 1 insert '567'·····································0001.00:00001(01)
```

The string 567 is to be inserted in line 1.00 as of column 5. Consequently, no characters are overwritten in line 1.00.

```
1.00 123456780<·····························································
2.00 348<···································································
3.00 ·····································································
```

9.21 @COMPARE (format 1) – Compare two work files

This format of the @COMPARE statement causes EDT to compare all or part of two work files with one another. The results of the comparison can be sent to a work file, SYSOUT or SYSLST as required.

Operation	Operands	F mode / L mode
@COMPARE	[procnr1] :lines1 **W**ITH [procnr2] :lines2 [,[int1] [(int2)] [**L**IST [line [(inc)]]]]	

procnr1, procnr2
: Numbers of the two work files that are to be compared (0..22). It is also possible to compare different ranges in the same work file (procnr1 equals procnr2). If one of the files for comparison is work file 0 and if a file has been opened in it for real processing with @OPEN, format 2 then @COMPARE is rejected with the error message EDT4935. If procnr1 or procnr2 is not specified then the value of the current work file is used for the missing operand.

lines1, lines2
: Line ranges that are to be compared with one another. The lines1 operand defines the line range in the first work file (procnr1). The lines2 operand defines the line range in the second work file (procnr2). Neither of these line ranges may be empty as otherwise the statement is rejected with the error EDT4932.

int1, int2
: int1 and int2 can be used to determine how tolerant EDT is to be if it finds non-identical lines. If EDT does not find at least int2 consecutive lines that are identical in the two files after examining int1 lines then it aborts the comparison.

  In addition, int2 specifies how many consecutive lines in a work file must match the corresponding number of consecutive lines in the other work file before EDT considers the ranges consisting of these lines to be identical.

  The following applies to int1 and int2: int2 ≤ int1 ≤ 65535. The default value for int1 is 10, and for int2 it is 1.

LIST — Specifies where EDT is to output the result of the comparison.

If LIST is specified without line, EDT outputs the result of the comparison to SYSLST. In this case, EDT outputs the line number and the first 51 characters of line content for every line for which no match is found.

If LIST is specified with line then EDT writes the result of the comparison to the current work file unless this is one of the two files involved in the comparison. Otherwise, the @COMPARE statement is rejected with the message EDT4909. The line number assignment can be influenced using the line and inc operands (see below). Only the numbers of the lines for which no match is found are output. The line content is not output.

If LIST is not specified, EDT outputs the result of the comparison to SYSOUT in interactive mode and to SYSLST in batch mode. The output format is the same as for output to the current work file.

line — The number of the line in the current work file which is to contain the first line of the result of the comparison. The format in which EDT writes the result is the same as that used when output is sent to SYSOUT.

inc — Increment used to form the line numbers which follow line. If inc is not specified then the increment implicitly specified by line is used (see section “Implicit increment assignment” on page 35).

Before performing the comparison, EDT internally converts each line into UTF16 and compares the resulting lines as byte sequences. The lines are identical if both the line content and line length of this byte sequence are identical. The line numbers are ignored during the comparison of the files. If both work files use the same character set then this procedure is equivalent to a byte-by-byte comparison of the original lines.

It may be necessary to convert the output of the result of this comparison into a suitable character set. If the output is sent to SYSOUT or SYSLST then this is the character set that has been defined for SYSOUT or SYSLST. If the output is sent to the current work file then this is the character set defined for this work file. If no character set is defined for the current work file then output takes place using the character set of the compared work files. If these use different character sets then the output takes place in the character set UTFE.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

EDT starts the comparison at the beginning of the specified line ranges. If EDT identifies a non-identical pair of lines then it reads forwards in the two files to try to find the next block of int2 identical lines.

In this case, EDT reads a maximum of int1 lines in each file. If, in this range, EDT identifies int2 consecutive identical lines, it aligns the two files for further comparison at this point. Otherwise, it aborts the comparison.

By choosing a suitable setting for `int2`, it is also possible to cause EDT to consider multiple consecutive lines as a unit during the comparison. This can be of use in address files if the address specifications consist of multiple lines (see example 2).

If the same values are selected for `int2` and `int1` then EDT will not be able to find any matching line ranges in the two files if even a single pair of lines is not identical.

EDT reports the result of the comparison in the form of commented lists of line numbers.

In this case, it does not separately specify whether lines or line ranges are identical. The ranges for which a matching range has been found in the other file are identified together by EDT (provided that they contain at least `int2` lines), i.e. they form pairs of matching ranges. If empty ranges are permitted then it is also possible to form pairs for the non-matching ranges since two adjacent pairs of matching ranges are necessarily separated by precisely one pair of non-matching ranges, one of which may be empty.

EDT generates a commented list for each pair of non-matching ranges as a function of their size and location.

If both members of a pair of non-matching ranges are not empty but comprise no more than `int1` - `int2` lines then the ranges are considered to be different:

```
NON-MATCHING LINES
  ln       ln
   .        .
   .        .
   .       ln
  ln
```

If both members of a pair of non-matching ranges are not empty and one range contains more than `int1 – int2` lines then the output of the above list is shortened accordingly and the comparison is aborted with the message

```
NOTHING SEEMS TO MATCH
```

If one member of a pair of non-matching ranges is empty and the other contains no more than `int1` - `int2` lines then the non-empty range is listed as an additional range:

```
EXTRA LINES IN 1ST FILE
  ln
   .
   .
   .
  ln
```

If the non-empty range is located in the second file then the output is equivalent but has a different heading:

```
EXTRA LINES IN 2ND FILE
```

If the non-empty range contains more than `int1 - int2` lines and is not located at the end of the line range for comparison then the output of the above list is shortened accordingly and the comparison is aborted with the message

```
NOTHING SEEMS TO MATCH
```

If the non-empty range of more than int1 - int2 lines is located at the end of the line range for comparison then the message

```
REACHED LIMIT ON 1ST FILE
```

or

```
REACHED LIMIT ON 2ND FILE
```

Is output instead, where `1ST` or `2ND` designates the file which contains the empty range.

If no non-matching ranges of more than `int1 - int2` lines are found in either file up to the end of the line range for comparison then the note

```
REACHED LIMIT ON BOTH FILES
```

is also output if a pair of non-matching ranges is located at the end of the line ranges for comparison. In contrast, if a pair of matching ranges is located at the end of the line ranges for comparison, the message

```
REACHED LIMIT ON BOTH FILES AT SAME TIME
```

is output. If both of the line ranges for comparison match fully then this is the only message to be output.

Example 1

```
   1.     @PROC 1
   1.     @COPY FILE=PROC-FILE.1 ---------------------------------------- (1)
   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.     @COPY FILE=PROC-FILE.2 ---------------------------------------- (2)
   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) ------------------------------- (3)
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) ------------------------------- (4)
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) ------------------------------- (5)
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.
```

(1) The SAM file `PROC-FILE.1` is read into work file 1.

(2) The SAM file `PROC-FILE.2` is read into work file 2.

(3) If the examination of 5 lines in each of the two files does not reveal at least 2 consecutive identical line pairs then the comparison is to be aborted. The comparison is continued through to the end of the two files.

(4) The @COMPARE issued in (3) is issued again in slightly modified form. It is now necessary to find at least 3 consecutive identical line pairs. This time, EDT aborts the comparison.

(5) The @COMPARE issued in (4) is issued again in slightly modified form. The comparison should now only be aborted after 6 lines have been examined. It is continued through to the end.

Example 2

This example assumes that the work files have already been filled with the corresponding data.

```
   5.     @PROC 1
  21.     @PRINT ------------------------------------------------------ (1)
   1.0000 Donald Duck
   2.0000 Am Dorfteich 11
   3.0000 12345 Entenhausen
   4.0000 -
   5.0000 Dagobert Duck
   6.0000 Schlossallee 1a
   7.0000 12345 Entenhausen
   8.0000 -
   9.0000 Daisy Duck
  10.0000 Am Dorfteich 12
  11.0000 12345 Entenhausen
  12.0000 -
  13.0000 Gustav Gans
  14.0000 Im Wiesengrund 10
  15.0000 12345 Entenhausen
  16.0000 -
  17.0000 Gustav Gans
  18.0000 Schmale Gasse 7
  19.0000 12345 Entenhausen
  20.0000 -
```

```
  21.     @END
   5.     @PRINT ------------------------------------------------------ (2)
   1.0000 Gustav Gans
   2.0000 Im Wiesengrund 10
   3.0000 12345 Entenhausen
   4.0000 -
   5.     @COMPARE 0:& WITH 1:&, 9999(4) ------------------------------- (3)
EXTRA LINES IN 2ND FILE
               1.0000
               2.0000
               3.0000
               4.0000
               5.0000
               6.0000
               7.0000
               8.0000
               9.0000
              10.0000
              11.0000
              12.0000
EXTRA LINES IN 2ND FILE
              17.0000
              18.0000
              19.0000
              20.0000
REACHED LIMIT ON BOTH FILES
    5.    @ON 2 CHANGE '10' TO '13'
    5.    @COMPARE 0:& WITH 1:&, 9999(4) ------------------------------- (4)
NON-MATCHING LINES
   1.0000      1.0000
   2.0000      2.0000
   3.0000      3.0000
   4.0000      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
              17.0000
              18.0000
              19.0000
```

```
             20.0000
REACHED LIMIT ON BOTH FILES
    5.
```

(1) Work file 1 is output. This is an address file.

(2) On return from work file 1, work file 0 is output. It contains an address (4 lines) which is to be searched for in work file 1.

(3) The search is performed by comparing the two files. Selecting the value `9999` for `int1` (see the operand description) ensures that the comparison is not aborted. Selecting `4` for `int2` ensures that only ranges with `4` matching lines are considered to be identical. The output identifies the presence of `EXTRA LINES` both before and after the matching range, i.e. the address is present in work file 1.

(4) Once the building number has been changed, the search is repeated. Since no further range of `4` matching lines is found, only `NON-MATCHING LINES` are reported. The address is therefore not present.

If the comparison is performed with @COMPARE 0:& WITH 1:&, 9999(1) instead of with @COMPARE 0:& WITH 1:&, 9999(4) then the same output is obtained for both matching and non-matching building numbers since EDT resynchronizes on the first matching line (`12345 Entenhausen`).

9.22 @COMPARE (format 2) – Compare two work files line by line

Format 2 of the @COMPARE statement can be used to compare the contents of two work files line by line. EDT stores the results in a work file. This is deleted before the result is stored in it. It is also possible to send the results to SYSLST and, in L mode, SYSOUT.

Operation	Operands	F mode / L mode
@COMPARE	{ [procnr1] **W**ITH procnr2 / procnr1 } [**L**IST [procnr3]] [,procnr4]	

procnr1 Number of the work file that is to be compared. If procnr1 is not specified then the current work file is compared with procnr2.

procnr2 Number of the work file against which the comparison is to be performed. If procnr2 is not specified then procnr1 is compared with the current work file.

LIST If LIST is specified then the result is stored in work file procnr3. If procnr3 is not specified then the result is output to SYSLST.

If LIST is not specified then in the interactive mode's L mode, the result is output to SYSOUT, in batch mode it is output to SYSLST and in F mode it is written to work file 9. Work file 9 is deleted before being used. If a file is open in work file 9 then the message EDT5189 is output and the statement is not executed.

procnr3 Work file in which the detailed result of the comparison is stored if any such result is generated (see below). Line numbers are assigned using the procedure “Insertion between two lines” (see section “Line number assignment” on page 36).

The work file is deleted before being used. If a file is open in this work file then it is implicitly closed without being written back (@CLOSE NOWRITE).

procnr4 The specification of a work file as an auxiliary file is now only permitted for reasons of compatibility. Any work file specified here is not used by EDT.

The work files procnr1 and procnr2 must be different from one another. Otherwise, the @COMPARE statement is rejected with the message EDT5499. The work file procnr3 can be identical to procnr1 or procnr2. However, in this case no detailed result is output (see below).

If all the lines to be compared are either identical or different then only the message EDT0291 or EDT0290 respectively is output. In this case, no detailed result of the comparison is output.

If a detailed result is to be sent to procnr3 then the message EDT0297 is output once the comparison has been completed. In this case, if one of the two files for comparison is the work file which is to contain the result then the message EDT5350 is output and no detailed result is output.

If one of the files for comparison is work file 0 then no ISAM file may be opened for real processing with @OPEN, format 2. Otherwise, the @COMPARE statement is rejected with the message EDT4935.

To make it possible to query the result of the comparison in EDT procedures, the EDT error switch is set in addition to the output of the messages EDT0290 and EDT0297. This can be queried using the @IF statement (see @IF statement):

	EDT error switch	**Work file procnr3**
EDT0291	Not set	Empty
EDT0290	Set	Empty
EDT0297	Set	Not empty

If it is necessary to distinguish between all the cases listed above then it is necessary to reset the EDT error switch with @RESET and delete the work file procnr3 before performing a comparison with @COMPARE.

Before performing the comparison, EDT internally converts each line into UTF16 and compares the resulting lines as byte sequences. The lines are identical if both the line content and line length of these byte sequences are identical. The line numbers are ignored during the comparison of the files. If both work files use the same character set then this procedure is equivalent to a byte-by-byte comparison of the original lines.

It may be necessary to convert the output of the result of this comparison into a suitable character set. If the output is sent to SYSOUT or SYSLST then this is the character set that has been defined for SYSOUT or SYSLST. If the output is sent to a work file then this is the character set of the compared work files. If these use different character sets then the output takes place in the character set UTFE.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

The format of the output is identical irrespective of whether it is written to a work file or sent to SYSLST or SYSOUT:

```
LINE#( 1)          FILENAME: DAT.270104
         LINE#( 0) FILENAME: COMP.1
```

A header line is output which identifies the columns assigned to the compared work files together with LINE#... and the number of the relevant work file (in parentheses). In addition, if present, the name of an opened file or library element or a local @FILE entry is output.

```
0007.10  CUST-100     SORT
0007.20  CUST-200     PERCON
        0007.30  CUST-700     FDDRL

0010.00          $CUST-900    LMS
        0010.00  $CUST-900    LMSCONV
```

In the case of lines which occur in only one work file, the line numbers and the content of the records (possibly truncated by 17 characters) are output. Here, the location of the line in either column 1 or in column 2 under the heading LINE#... indicates which of the work files contains the record. This also applies equivalently for records with different contents. These occur with one content only in the first work file and with the other content only in the second work file and will generally appear consecutively.

```
0008.00=0010.00
0018.00=0020.00
```

In the case of lines with the same content, the identified line numbers are output in the form 0001.00=0006.00. If a number of consecutive records are identical (range of identical records) then only the first and last pairs of line numbers in the range are output (for further details, see the example).

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 ·····································································
:                                                                       :
:                                                                       :
:                                                                       :
1·····························································0001.00:00001(02)
```

Processing switches 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 ·····································································
:                                                                       :
:                                                                       :
:                                                                       :
@compare 2 with 1 list 3;  3·····································0001.00:00001(01)
```

Work file 2 is compared with work file 1 and the result is stored in work file 3. Processing then switches 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
·····························································0000.10:00001(03)
```

The result of comparing work files 1 and 2 is stored in work file 3.

9.23 @CONTINUE – Empty statement

The @CONTINUE statement does not perform any action. It is used to generate a line in EDT procedures which can be branched to by means of a @GOTO statement. It can also be used to insert comments in EDT procedures. The @NOTE statement has the same functionality as @CONTINUE.

Operation	Operands	L mode
@CONTINUE	[comment]	

comment The `comment` operand may contain any text as a comment.

Alongside the insertion of comments, this statement is also frequently used to define a last line in an EDT procedure which can be specified as the destination of a branch operation in a @GOTO or an @IF statement. This construction is required if an EDT procedure is called in an external loop with a loop counter (e.g. @DO 5,!=%,$), and an @IF ... RETURN would result in an unwanted abort of the external loop. Instead, processing branches 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' ---------------------- (1)
1.01   @ @CON          DISPLAY IT ON THE SCREEN
1.02   @ @ON ! FIND 'W'
1.03   @ @IF .FALSE. : @GOTO 2
1.04   @ @PRINT !
1.05   @2.00
2.00   @ @CONTINUE ------------------------------------------------- (2)
2.01   @END
6.     @DO 1,!=1,$ ------------------------------------------------- (3)
1.0000 WITH EDT
2.0000 ANYONE WHO KNOWS
4.0000 WRITE HIS PROGRAM
6.
```

(1) In this case, @CONTINUE is used to insert a comment.

(2) In this case, @CONTINUE is required because there must be a last line in a procedure that can be branched to.

(3) @DO with a loop counter executes the procedure in work file 1 which acts on work file 0.

9.24 @CONVERT – Convert uppercase or lowercase

The @CONVERT statement is used to convert lowercase characters into uppercase or uppercase characters into lowercase in line ranges.

Operation	Operands	F mode, L mode
@CONVERT	[[{ lines / svars } [,...]] **T**O =] { **UP**PER / **LOW**ER }	

lines — One or more line ranges in which conversion is to be performed. Only existing lines are processed.

svars — One or more ranges of string variables in which conversion is to be performed.

UPPER — All lowercase characters are converted into uppercase.

LOWER — All uppercase characters are converted into lowercase.

Either `UPPER` or `LOWER` must be specified. If neither `lines` nor `svars` is specified then conversion is performed throughout the entire current work file.

The information about which characters in the character set used in the current work file are lowercase or uppercase characters is obtained from XHCS. All other characters are left unchanged.

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

9.25 @COPY (format 1) – Read in a file

@COPY (format 1) is used to read an existing file in full into the current work file. The work file does not have to be empty when this is done. It is possible to specify the position in the work file at which the file is to be inserted. After being read in, the file is closed again.

Whenever this section refers to a “file”, this can be a SAM file, an ISAM file, a library element or a POSIX file.

Operation	Operands	F mode, L mode
@COPY	**L**IBRARY=path1 ([**E**LEMENT=] elname [(vers)][,eltype]) **E**LEMENT=elname [(vers)][,eltype] **F**ILE = { path2 / *linkname } [,**K**EY = { **L**INENUMBER / **D**ATA / **I**GNORE }] **P**OSIX - FILE = xpath [,**C**ODE = name] [[,] { **B**EFORE / **A**FTER } line]	

LIBRARY= A library element is to be read in. This is defined by explicitly specifying the library name and the element designation.

path1 Name of the library.

elname Name of the element.

vers Version of the required element (see the LMS User Guide [14]). If `vers` is not specified or if `*STD` is specified then the highest available version of the element is selected.

eltype Type of element. Permitted type specifications are `S`,`M`, `P`, `J`, `D`, `X`, `*STD` as well as freely selectable type names having one of these types as basic type. If `eltype` is not specified then the default type specified with @PAR ELEMENT-TYPE is used. The permitted element types and their meanings are described in chapter “File processing” on page 131.

ELEMENT= A library element is to be read in. This is defined by means of the element designation without any library name specification. The default library set with @PAR LIBRARY is used implicitly (if @PAR LIBRARY has been specified, otherwise the error message EDT5181 is issued).

The operands elname, vers and eltype have the same meaning as when a library is specified explicitly (see above).

FILE= A BS2000 file is to be read in.

path2 Name of the BS2000 file (fully qualified file name) that is to be read in.

*linkname File link name of the BS2000 file that is to be read in. The file name and the file attributes are stored in the Task File Table. The file link name must not be specified as the special file name *BY-PROGRAM. This results in the error EDT4923. If no file link name is defined then the statement is rejected with the message EDT5480.
If the file link name is declared as the special file name *DUMMY then it is treated as an existing empty file.

KEY= In the case of ISAM files, specifies the location at which the ISAM key is stored in the work file. In the case of other file types, this operand is ignored.

LINENUMBER
The ISAM key is stored as a line number in the work file. Any existing lines with the same line numbers are overwritten. The operands BEFORE and AFTER may not be specified. If the ISAM key cannot be interpreted as a line number because the position of the key differs from the default value, the key is too long or the keys are not numerical then the message EDT5459 is output and the file is not read in.

DATA The ISAM key becomes a component of the data range in the work file.

IGNORE The ISAM key is not stored in the work file. This is the default value. If the position of the key differs from the default value, the message EDT5466 is output and the file is not read in.

POSIX-FILE= A POSIX file is to be read in.

xpath Path name of the POSIX file that is to be read into the current work file. The xpath operand can also be specified as a string variable. It *must* be specified as a string variable if the path name contains characters which have a special meaning in EDT syntax (e.g. blanks, semicolons in F mode or commas).

CODE= Defines the character set that is to be assumed for the POSIX file. Since it is not possible to assign character sets to POSIX files in the POSIX file system, a user specification is required here.

If CODE is not specified then the character set defined in @PAR CODE is assumed.

name Character set of the POSIX file that is to be read in. The name of a valid character set must be specified for name (see section “Character sets” on page 47).

EBCDIC The keyword EBCDIC is now only supported for reasons of compatibility and is a synonym for the character set EDF041.

ISO The keyword ISO is now only supported for reasons of compatibility and is a synonym for the character set ISO88591.

BEFORE The file is inserted in front of the specified line in the work file. This operand may not be specified if KEY=LINENUMBER is defined.

AFTER The file is inserted after the specified line in the work file. This operand may not be specified if KEY=LINENUMBER is defined.

line Line number before or after which the file is inserted.

If the specified file does not exist or cannot be accessed as required or if the file cannot be read in successfully then the statement is rejected with a corresponding error message.

In neither BEFORE nor AFTER is specified and if the KEY operand is not equal to LINENUMBER then the file is inserted after the last line of the current work file.

When ISAM files are read with the operand KEY=LINENUMBER, the line numbers in the work file are derived from the file's ISAM key. In all other cases, they are formed using the procedure “Insertion between two lines” (see section “Line number assignment” on page 36).

If the current work file is empty and has the character set *NONE then it is assigned the character set of the file that is to be read in. If this character set is *NONE then the work file is assigned the character set EDF03IRV.

If the work file already has a character set then the records that are to be read in are converted from the file's character set into the work file's character set. If the file that is to be read in contains characters which cannot be displayed in the work file's character set then these are replaced by a substitute character if such a character has been specified. (see @PAR SUBSTITUTION-CHARACTER), otherwise the file is not read in and the error message EDT5453 is output.

If the file is present in a Unicode character set and contains an illegal byte sequence, e.g. surrogate characters, then it will be impossible to read it even if SUBSTITUTION-CHARACTERS is specified. In this case, the read operation is rejected with the message EDT5454.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
@COPY LIBRARY=MACLIB(ELEMENT=XYZ,M) AFTER 12.3
```

The element with the name XYZ, the highest existing version and the element type M (macro) from the library MACLIB is inserted in full after line 0012.3000 in the current work file.

```
@PAR LIBRARY=DATA
@COPY ELEMENT=PERSONAL(@),D
```

The element with the name PERSONAL, the highest existing version and the element type D (text data) from the library DATA is inserted after the last line in the current work file.

```
@COPY FILE=FILE.ISAM,KEY=LINENUMBER
```

The ISAM file FILE.ISAM is read into the work file. The line numbers in the work file are formed from the ISAM key. Any existing lines are overwritten.

```
@COPY POSIX-FILE=/home/user1/test/data,CODE=UTF8
```

The POSIX file data in the directory /home/user1/test with the character set UTF8 is inserted after the end of the current work file. When this is done, the file is converted into the work file's character set.

9.26 @COPY (format 2) – Copy lines or string variables

The @COPY statement copies records from the current or another work file or the content of a string variable into the current work file.

For the sake of clarity, the line range in the source work file which contains the records that are to be copied or the range of string variables are referred to as the “source range” below. The line range in the current work file into which the records from the source work file are to be copied is referred to as the “target range”.

Operation	Operands	F mode, L mode
@COPY	{ { lines [(procnr)] / svars } [**T**O {line1 [(inc)] [:] [line2]} [,...]] } [,...]	

lines — Contiguous line range that is to be copied into the current work file. Symbolic line numbers in `lines` refer to the line numbers of the current work file even if the lines are copied from another work file.

procnr — Number of the source work file from which the lines are to be copied (`0..22`). If `procnr` is not specified then the lines are copied from the current work file. An active work file may not be specified.
If the `TO` operand is not specified then `procnr` must not be the current work file.

svars — Range of string variables whose contents are to be copied into the current work file.

TO... — The operands which follow `TO` define the target range or ranges. If no target range is specified then the line numbers in the source work file are taken over into the current work file.
If the source work file is the current work file or if string variables are copied then `TO...` must be specified. In these cases, if no target range is specified then the @ COPY statement is rejected with the error message `EDT3218`.

line1 — Number of the first line in the target range.

inc — Increment used to form the line numbers following `line1`. If `inc` is not specified then the increment implicitly specified by `line1` is used (see section “Implicit increment assignment” on page 35).

: — The operands `line1` and `line2` should be separated by : if `inc` is not specified.

line2 Specifies the largest possible line number in the target range up to which the copying of records is permitted.

As a result, nothing is copied into lines in the current work file with line numbers higher than `line2`. This also applies if it is not possible to copy all the records in the source range to the target range.
If `line2` is not specified then the @COPY statement does not define any maximum value for the line numbers in the target range.

In the @COPY statement, it is possible to specify multiple comma-separated source ranges each of which are associated with multiple target ranges. The number of source and target ranges is only limited by the maximum permitted length of EDT statements.

If the source and target ranges overlap then the source range is copied line-by-line. This means that a record may initially be copied to a line and then be copied again from this line if the line is present in both the source and target ranges. In this way, it is possible to create multiple copies of the source range or parts of it in the target range.

Any existing lines with the same line numbers present in the work file are overwritten on the copy operation.

If a line with a number greater than the previous highest line number is created then the current line number is modified.

If the current work file is empty and has the character set `*NONE` then it is assigned the character set of the source work file or the first specified string variable when the copy operation is performed. If the current work file has a character set then the lines to be copied or the contents of the string variables are converted into the character set of the current work file. If characters which cannot be displayed in the work file's character set are identified then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @ COPY statement is rejected and error message `EDT5453` is output.

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

Note

Since the above syntax permits the omission of the `TO` operand, it is not always possible to distinguish unambiguously between the target and source ranges. In such cases, EDT interprets the ambiguous specification as a target range. Thus, for example, in the input

```
@COPY 2-3(1) TO 7,1(1)
```

the specification `1(1)` is interpreted as a second target range (the `1` in parentheses is interpreted as the increment), whereas the specification `1(0)` at this point would be interpreted as the next source range (the `0` cannot be an increment and is interpreted as a work file number). If, in this example, the user wants to force the specification to be interpreted as a source range, it would be possible, for example, to enter

```
@COPY 2-3(1) TO 7,1-1(1)
```

to eliminate all ambiguities.

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)··············0001.00:00001(00)
```

The three @COPY statements are intended to copy line 1 to line 7, line 2 to line 5 and line range from 1 to 3 to the line range starting at line 30.1 with the explicit increment 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

The @COPY statement can be used to duplicate line ranges one or more times if the send and receive areas overlap. In the example below, the first line 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·····························································0001.00:00001(00)
```

This statement copies the line range from line number 1 to 2 to the line range starting at line number 1.5 with the implicit increment 0.1.

In this case, EDT starts by copying line 1 to line 1.5. This line is located in the specified source range. Consequently line 1.5 is copied to line 1.6 with the implicit increment 0.1. Accordingly, line 1.6 is copied to 1.7, ... , line 1.9 to 2.0 (and the content of line 2 is overwritten) and line 2.0 is then copied to 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·····························································0001.00:00001(00)
```

The line range 3 to 5 is to be copied to the line range 4.1 to 5 with the implicit increment 0.1.

To do this, EDT first copies line 3 to line 4.1 and line 4 to line 4.2. The two newly created lines are located in the specified source range. As a result, line 4.1 is copied to line 4.3, line 4.2 to 4.4 ... , line 4.8 to 5.0. This operation overwrites the content of line 5. Lines 4.9 and 5.0 are not copied since the highest possible line number in 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 ·····························································
·····························0001.00:00001(00)
```

9.27 @CREATE (format 1) – Check line

Format 1 of the @CREATE statement creates a line with the specified content.

Operation	Operands	F mode, L mode
@**CR**EATE	line [:] [string[,...]] [,**C**ODE=name]	

line — The line number in the current work file that is to be created. If this line already exists then it is completely overwritten.

: — This must be specified if `line` cannot be unambiguously separated from `string`.

string — One or more strings which are to be joined in the specified order and inserted as a line.

name — Character set that is to be defined for the current work file if this is empty and has the character set `*NONE`.

During the first step, the character strings specified in `string` are joined to one another. If all the strings involved have the same character set then the intermediate result is also assigned this character set. If the involved strings have different character sets then the intermediate result is assigned the character set `UTFE`.

If, after conversion, the intermediate result exceeds the maximum length of 32768 characters then it is truncated to the maximum length and message `EDT2400` is output.

If the `CODE` operand is not specified and the current work file is empty and has the character set `*NONE` then the intermediate result is inserted in the line without being converted. The character set used for the intermediate result is assigned to the work file.

If the `CODE` operand is specified and the current work file is empty and has the character set `*NONE` then the intermediate result is converted into the character set `name` before being inserted. Precisely this character set is then defined for the current work file.

If the `CODE` operand is specified and the current work file already has a different character set then the character set `*NONE` then the @CREATE statement is not executed and the message `EDT5458` is issued.

If the `CODE` operand is not specified and the current work file already has a character set then the intermediate result is converted into the work file's character set before being inserted.

If the string that is to be inserted contains characters which cannot be displayed in the work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @CREATE statement is rejected and error message `EDT5453` is output.

If `string` is not specified then the line is created as an empty line (line of length 0). The character set `EDF041` is defined for the current work file if this is empty and has the character set `*NONE` and the `CODE` operand has not been specified.

The @CREATE statement does not modify the current line number. This also applies if new lines are created after the end of the existing work file.

Example

```
   1.00 THIS IS THE FIRST LINE<·····················································
   2.00 THIS IS THE SECOND LINE<····················································
   3.00 ····························································································
create 3 'LINE 3 IS CREATED WITH @CREATE'·····················0001.00:00001(00)
```

A new line 3 is created with @CREATE.

```
   1.00 THIS IS THE FIRST LINE<·····················································
   2.00 THIS IS THE SECOND LINE<····················································
   3.00 LINE 3 IS CREATED WITH @CREATE<·····························································
   4.00 ····························································································
create 3:3, ' AND GETS LONGER'·····························0001.00:00001(00)
```

The new line 3 is created from the content of the old line 3 joined to the new text `AND GETS LONGER`

```
   1.00 THIS IS THE FIRST LINE<·····················································
   2.00 THIS IS THE SECOND LINE<····················································
   3.00 LINE 3 IS CREATED WITH @CREATE AND GETS LONGER<·····························
   4.00 ····························································································
create 4:1, ' CHAINED WITH ',2,1; edit long on···············0001.00:00001(00)
```

The new line 4 is created and consists of the joining of line 1, the text `CHAINED WITH` and lines 2 and 1.

`EDIT LONG ON` is input to make it possible to display the content of line 4 in full at the terminal.

```
THIS IS THE FIRST LINE<·····················································
THIS IS THE SECOND LINE<····················································
LINE 3 IS CREATEDE WITH @CREATE AND GETS LONGER<·····························
THIS IS THE FIRST LINE CHAINED WITH THIS IS THE SECOND LINETHIS IS THE FIRST LIN
E<·····························································································
:                                                                             :
:                                                                             :
:                                                                             :
create 4:1:1-12:,'FOURTH',2:19-23: ; index on ·················0001.00:00001(00)
```

A new line 4 is created. The new content of this line consists of joining columns 1 to 12 of line 1, the word `FOURTH` and columns 19 to 23 of line 2 in this order.

The work window then switches back to the default format.

```
   1.00 THIS IS THE FIRST LINE<··································
   2.00 THIS IS THE SECOND LINE<·································
   3.00 LINE 3 IS CREATED WITH @CREATE AND GETS LONGER<·········
   4.00 THIS IS THE FOURTH LINE<·································
   5.00 ···························································
```

9.28 @CREATE (format 2) – Assign string to string variable

Format 2 of the @CREATE statement is used to assign strings to string variables.

Operation	Operands	F mode, L mode
@**CR**EATE	svarex [:] [string[,...]] [,**C**ODE=name]	

svarex — New string variable that is to be created.

: — This must be specified if `svarex` cannot be unambiguously separated from `string`.

string — One or more strings which are to be chained together in the specified order and assigned to a string variable.

name — Character set that is to be defined for the specified string variable.

During the first step, the character strings specified in `string` are chained together. If all the strings involved have the same character set then the intermediate result is also assigned this character set. If the involved strings have different character sets then the intermediate result is assigned the character set `UTFE`.

If, after conversion, the intermediate result exceeds the maximum length of 32768 characters then it is truncated to the maximum length and message `EDT2400` is output.

If the `CODE` operand is not specified then the content and character set of the intermediate result are assigned to the string variable.

If the `CODE` operand is specified then this character set is assigned to the string variable and intermediate result is converted into the character set `name` before being assigned to the string variable. If the string that is to be inserted contains characters which cannot be displayed in the character set specified in `name` then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @CREATE statement is rejected and error message `EDT5453` is output.

If neither the string nor the `CODE` operand is specified then the string variable is created with a blank and the character set `EDF041`. If `string` is not specified but the `CODE` operand is then the string variable is created with a blank in the character set specified in the `CODE` operand.

Example

```
@READ 'SRC.EDF041' ------------------------------------------------------ (1)
@CREATE #S01:1,CODE=UTF16 ----------------------------------------------- (2)
@DELETE
@READ 'SRC.EDF045' ------------------------------------------------------ (3)
@CREATE #S02:3,CODE=UTF16 ----------------------------------------------- (4)
@CREATE #S03: #S01,#S02 ------------------------------------------------- (5)
```

(1) The character set EDF041 is defined for the work file.

(2) The CODE operand is used to define the character set UTF16 for the string variable #S01. The first line in the work file is converted from EDF041 to UTF16 and assigned to the string variable #S01.

(3) The character set EDF045 is defined for the work file.

(4) The CODE operand is used to define the character set UTF16 for the string variable #S02. The third line in the work file is converted from EDF045 to UTF16 and assigned to the string variable #S02.

(5) The character set UTF16, which is the character set of the strings #S01 and #S02, is implicitly defined for the string variable #S03. The contents of the string variables #S01 and #S02 are chained together and assigned to the string variable #S03.

9.29 @CREATE (format 3) – Read in string and create line

Format 3 of the @CREATE statement is used to read a string from the terminal or from SYSDTA and create a line with its content.

Operation	Operands	L mode
@**CR**EATE	line **R**EAD [string[,...]] [,**C**ODE=name]	

line — The line number in the current work file that is to be inserted in a string. If this line already exists then it is completely overwritten.

string — One or more strings which are to be chained together in the specified order and output at the terminal as a prompt.

If string is not specified then no prompt is output at the terminal.

name — Character set that is to be defined for the current work file if this is empty and has the character set *NONE.

In interactive mode, the prompt formed from the operands is output at the terminal and a string is read. If the prompt formed from the operands exceeds the maximum length of 32763 bytes then it is truncated to the maximum length and error message EDT2402 is output. If string is not specified then a string is read from SYSDTA instead of from the terminal.

In batch mode, string is ignored and the string is always read from SYSDTA.

The maximum length of the read string depends on the input medium.

If the current work file already has a character set then the read string is converted into this character set before being inserted. If the string that is to be inserted contains characters which cannot be displayed in the work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @CREATE statement is rejected and error message EDT5453 is output.

If the CODE operand is not specified and the current work file is empty and has the character set *NONE then the read string is inserted in the line without being converted. The communications character set is defined as the character set for the current work file.

If the CODE operand is specified and the current work file already has a different character set then the character set then the @CREATE statement is not executed and the message EDT5458 is issued.

Entering F1 without text at a terminal causes the specified line to be created as an empty line (line of length 0). Empty input that is sent with DUE or another function key is ignored and the prompt is output again.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
   1.     @PROC 1
   1.     @CREATE 1 READ '*** NAME OF THE FIRST PARTICIPANT ?',CODE=UTF16 -
(1)
*** NAME OF THE FIRST PARTICIPANT ? SCHÖLLER
   1.     @PROC 2 ------------------------------------------------------ (2)
   4.     @PRINT
   1.0000 MAIER
   2.0000 LINE IS OVERWRITTEN
   3.0000 SCHMIDT
   4.     @CREATE 2 READ '*** MUELLER OR MÜLLER ?' ---------------------- (3)
*** MUELLER OR MÜLLER ? MÜLLER
```

(1) The CODE operand is used to define the character set UTF16 for work file 1 which is empty and has the character set *NONE. The prompt '*** NAME OF THE FIRST PARTICIPANT ?' is output at the terminal and a string is read. This is converted into the character set UTF16 and written to the first line of the work file.

(2) The character set EDF041 is defined for work file 2.

(3) The prompt '*** MUELLER OR MÜLLER ?' is output at the terminal and a string is read. This is converted into the work file's character set (EDF041) and written to the second line. When this is done, the existing content of the second line is completely overwritten.

9.30 @CREATE (format 4) – Read in line and assign to string variable

Format 4 of the @CREATE statement is used to read a string from the terminal or from SYSDTA and assign it to a string variable.

Operation	Operands	L mode
@CREATE	svarex **R**EAD [string[,...]] [,**C**ODE = name]	

svarex — New string variable that is to be created.

string — One or more strings which are to be chained together in the specified order and output at the terminal as a prompt.

If string is not specified then no prompt is output at the terminal.

name — Character set that is to be defined for the specified string variable.

In interactive mode, the prompt formed from the operands is output at the terminal and a string is read. If the prompt formed from the operands exceeds the maximum length of 32763 bytes then it is truncated to the maximum length and error message EDT2402 is output. If string is not specified then a string is read from SYSDTA instead of from the terminal.

In batch mode, string is ignored and the string is always read from SYSDTA.

The maximum length of the read string depends on the input medium.

If the CODE operand is not specified then the content of the string is assigned to the string variable and the communications character set is defined as its character set.

If the CODE operand is specified then this character set is assigned to the string variable and the read string is converted into the character set name before being assigned. If the string that is to be inserted contains characters which cannot be displayed in the character set specified in name then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @CREATE statement is rejected and error message EDT5453 is output.

Entering F1 without text at a terminal causes the specified string variable to be created as an empty string variable. Empty input that is sent with DUE or another function key is ignored and the prompt is output again.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example 1

```
   6.     @PRINT
   1.0000 HELLO
   2.0000 NO-ONE IS TO LEAVE
   3.0000 THE ROOM
   4.0000 LINE
   5.0000 IS TO BE
   6.     @SET #S1 = ' OUTPUT *** '
   6.     @PROC 1
   1.     @ @CREATE #S2 READ '*** WHICH ',4,5,#S1 ----------------------- (1)
   2.     @ @SET #L2 = SUBSTR #S2 ---------------------------------------- (2)
   3.     @ @PRINT #L2
   4.     @END
   6.     @DO 1
*** WHICH LINE IS TO BE OUTPUT *** 2 ------------------------------------ (3)
   2.0000 NO-ONE IS TO LEAVE
   6.
```

(1) @CREATE...READ is to be used to create the string variable #S2. First, however, the text resulting from *** WHICH is output at the terminal together with the contents of lines 4 and 5 and the string variable #S1.

(2) The input is interpreted and stored in the line number variable #L2.

(3) The query output via the terminal is answered.

Example 2

```
@CREATE #SO1 READ 'MUELLER OR MÜLLER ?'
```

The prompt 'MUELLER OR MÜLLER ?' is output at the terminal and a string is read. The content of the read string is assigned to the string variable and the communications character set is defined as its character set.

```
@CREATE #SO2 READ 'RESIDENT IN GÜNZBURG OR DONAUWÖRTH ?',CODE=EDF041
```

The character set EDF041 is defined for the string variable #SO2. The prompt RESIDENT IN GÜNZBURG OR DONAUWÖRTH ?' is output at the terminal and a string is read. This is converted into the character set EDF041 and assigned to the string variable #SO2.

9.31 @DELETE (format 1) – Copy lines and string variables

This format of the @DELETE statement is used to delete lines in the current work file or a range of string variables either in part or completely.

Operation	Operands	F mode, L mode
@DELETE	{ lines / svars } [:cols [:] [,...]	

lines The line range to be deleted.

svars The range of string variables whose contents are to be deleted.

cols Column range in the specified lines or string variables that is to be deleted.

If only one column number is specified then the remainder of the line or string variable is deleted as of this column. If the first column specification is greater than the length of the line or string variable then the line or string variable is ignored.

If there is no column specification then the entire line or string variable is deleted (see below).

Whenever a line range is deleted, only the specified records are removed. Furthermore, no “clean up” operations are performed even if the work file does not contain any more records after deletion.

Completely deleting a string variable restores this to the status which it had before it was first assigned a value, i.e. it contains precisely one blank in the initial character set `EDF041`.

If an ISAM file has been opened for real processing in work file 0 using @OPEN (format 2) then the corresponding range in the ISAM file is also deleted. However, the file's catalog entry is always retained even if the file contains no further files after deletion.

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

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 ·····································0001.00:00001(00)
```

The line range consisting of line numbers 1 to 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·····································0002.10:00001(00)
```

Columns 7 to 10 inclusive are to be deleted throughout the entire 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 ·····························································
·····························0002.10:00001(00)
```

The specified range has been deleted.

9.32 @DELETE (format 2) – Completely delete work files

This format of the @DELETE statement is used for the complete deletion of work files.

Operation	Operands	F mode, L mode
@DELETE	[**A**LL | (procnr[,...])]	

ALL All the work files are deleted in full.

procnr The work files with the specified numbers (`0..22`) are deleted in full.

If no operand is specified then the current work file is deleted in full.

In the F mode statement line, the abbreviation `D` without operands is rejected with an error message in order to prevent the accidental deletion of the current work file when `D` statement codes are entered.

If one of the specified work files is an active work file then the statement is rejected with the message `EDT5476`.

Completely deleting a work file makes it into an empty work file. In particular, not only are all the records deleted. In addition, the work file's character set is set to `*NONE` and other work file-specific settings are restored to their default values (see section “Work files” on page 27). Furthermore, a file that has been fully deleted (except for the current work file) is removed from the set of work files in use (see @PROC USED).

The work files are deleted without a confirmation query, irrespective of their content. If files are open in the work files that are to be deleted then these are implicitly closed without being written back. This also applies to files opened for real processing using @OPEN (format 2). The records they contain are therefore not deleted.

9.33 @DELETE (format 3) – Delete files and library elements

This format of the @DELETE statement can be used to delete files or elements in a library.

Operation	Operands	F mode, L mode
@DELETE	**L**IBRARY=path1 ([**E**LEMENT=] elname [(vers)][,eltype]) **E**LEMENT=elname [(vers)][,eltype] **F**ILE = path2 **P**OSIX-FILE = xpath	

LIBRARY= A library element is to be deleted.

path1 Name of the library.

elname Name of the element.

vers Version of the element that is to be deleted (see the LMS User Guide [14]). If `vers` is not specified or if `*STD` is specified then the highest available version of the element is deleted.

eltype Type of element. Permitted type specifications are `S`, `M`, `P`, `J`, `D`, `X`, `R`, `C`, `H`, `L`, `U`, `F`, `*STD` and freely selectable type names having one of these types as basic type. If `eltype` is not specified then the default type specified with @PAR ELEMENT-TYPE is used. The permitted element types and their meanings are described in section “File processing” on page 131.

ELEMENT= The library element to be deleted is defined by means of its name without any library name specification. The default library specified with @PAR LIBRARY is used implicitly (provided that @PAR LIBRARY has been specified – otherwise the error message `EDT5181` is output).
The operands `elname`, `vers` and `eltype` have the same meaning as when a library is specified explicitly (see above).

FILE= A BS2000 file is to be deleted.

path2 The fully qualified file name of a BS2000 file that you want to delete.

POSIX-FILE= A POSIX file is to be deleted.

xpath Path name of the POSIX file that you want to delete.

The `xpath` operand can also be specified via a string variable. It must be specified via a string variable if it contains special characters which have a special meaning in EDT syntax (e.g. blanks or semicolons in F mode).

If the specified file does not exist or cannot be accessed as required then the statement is rejected with a corresponding error message.

Example

```
@DELETE LIBRARY=PROGLIB(ELEMENT=TESTOLD(VER2))
```

Version `VER2` of the library element `TESTOLD` in the library `PROGLIB` and with element type `S` is deleted.

9.34 @DELETE (format 4) – Delete record marks

This format of the @DELETE statement is used to delete record marks (see section “Record marks” on page 45).

Operation	Operands	F mode, L mode
@DELETE	**MA**RK [m[,...]]	

m — One or more record marks (`1..9`) that are to be deleted in all the records in the current work file.

If `m` is not specified then all the record marks `1` to `9` are deleted in the records present in the current work file.

Record marks with a special function (marks `13`, `14`, `15`, see section “Record marks” on page 45) are not deleted.

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

9.35 @DELIMIT – Declare text delimiter characters

The @DELIMIT statement declares characters that act as text delimiters when searches are performed with @ON (see section “Delimiter characters” on page 81).

Operation	Operands	F mode, L mode
@DELIMIT	= [**R** | str1 | +|– str2]	

R — The blank ␣ and the characters `+.!*();-/,?:'="` are declared as text delimiter characters.

str1 — String containing all the characters that are to be declared as text delimiter characters.

str2 — String containing additional characters to be declared as text delimiter characters (`+`) or characters that are no longer to be declared as text delimiter characters (`-`).

If no operand is specified (@DELIMIT =) then no characters are to be used as text delimiter characters.

9.36 @DIALOG – Call screen dialog

The @DIALOG statement can be used to switch EDT to screen dialog mode when input is read from SYSDTA (usually in BS2000 system procedures or from the subroutine interface). In screen dialog, the preceding read operation is interrupted and EDT reads its input from the terminal in F mode (or in L mode after the entry of @EDIT). The screen dialog can be exited again with @HALT, @END, @RETURN or K1. EDT then continues the interrupted read operation.

Operation	Operands	F mode, L mode
@DIALOG		

The statement is ignored in F mode. If, in L mode, @DIALOG is entered from a medium other than SYSDTA (e.g. if statements are read from an EDT procedure or if input is read from the terminal with the line number as the prompt) or if it is called in batch mode then the statement is rejected with the error message EDT5400 or EDT4920.

If screen dialog mode is called from a BS2000 system procedure then the statements @SYSTEM without operands and @EDIT ONLY are prohibited and are rejected with the message EDT4976. In this case, the only way to switch to the operating system is to press K2 provided that the BS2000 system procedure has not been protected against this with the option INTERRUPT-ALLOWED=NO see section “Access protection” on page 99).

The screen dialog is terminated with @HALT, @END, @RETURN or K1. If @DIALOG is called via the subroutine interface or from a BS2000 procedure then processing continues with the statement that follows @DIALOG. If there are no further statements at the subroutine interface after @DIALOG then control passes to the calling program. If it is called from SYSDTA (after @EDIT ONLY) then the next input is requested from SYSDTA. In all cases, EDT remains loaded and all the EDT parameter settings remain as they were at the point screen dialog mode was exited.

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=(4,5) ------------------------------------------ (1)
/START-EDTU
@PROC 1 ---------------------------------------------------------------- (2)
@COPY FILE=&FILE1 ------------------------------------------------------ (3)
@PAR SCALE=ON ---------------------------------------------------------- (4)
@DIALOG ---------------------------------------------------------------- (5)
@SETF(1) --------------------------------------------------------------- (6)
@WRITE FILE=&FILE2 ----------------------------------------------------- (7)
@HALT ------------------------------------------------------------------ (8)
/MODIFY-JOB-SWITCHES OFF=(4,5)
/ASSIGN-SYSDTA TO-FILE=*PRIMARY
/END-PROCEDURE
```

(1) Job switch 5 is set before EDT is loaded. This sets L mode and the input is read from SYSDTA.

(2) Processing switches to work file 1.

(3) A file is to be read in. The file name is queried while the procedure is running.

(4) The display of the column counter is activated.

(5) EDT is to be switched to the F mode screen dialog and the work window is to be output on the screen. It is then possible to input all F and L mode statements in this dialog mode. The F mode screen dialog is terminated with @END, @HALT or @RETURN or K1 and processing continues with the statement following @DIALOG.

(6) Work file 1 is again set as the current work file. This is necessary because the user may have set a different work file in the F mode screen dialog.

(7) Work file 1 is written to a SAM file. The file name is queried while the procedure is running.

(8) EDT is terminated.

```
/call-procedure name=proc.dialog
%/BEGIN-PROCEDURE LOGGING=A,PARAMETERS=YES(PROCEDURE-PARAMETERS
=(&FILE1=,&FILE2=),ESCAPE-CHARACTER='&')
%/ASSIGN-SYSDTA TO-FILE=*SYSCMD
%/MODIFY-JOB-SWITCHES ON=(4,5)
%/START-EDTU
%PROC 1
%@COPY FILE=&FILE1
%&FILE1=xmpl.dialog
```

The procedure `PROC.DIALOG` is started. This requests the name of the file that is to be read in. EDT then switches to F mode screen dialog.

```
      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   1.00 EDT is the BS2000 file<·····································
   2.00 editor, used for the user-<·····································
   3.00 friendly creation and editing<·····································
   4.00 of BS2000 files in SAM and ISAM formats<·····································
   5.00 as well as text-like library<·····································
   6.00 elements and POSIX files<·····································
:                                                                              :
:                                                                              :
:                                                                              :
@halt·····························································0001.00:00001(01)
```

The column counter is displayed in accordance with the default values specified under (4). @HALT terminates the F mode screen dialog again and the procedure which was interrupted by @DIALOG continues.

The procedure then queries the name of the file to which the work file is to be written. Depending on the actions performed in the F mode screen dialog, other messages may be output.

```
%@SETF (1)
%@WRITE FILE=&FILE2
%&FILE2=xmpl.dialog1
%@WRITE FILE=XMPL.DIALOG1
%@HALT
%/MODIFY-JOB-SWITCHES OFF=(4,5)
%/ASSIGN-SYSDTA TO-FILE=*PRIMARY
%/END-PROCEDURE
/
```

9.37 @DO (format 1) – Start EDT procedures from work files

Format 1 of the @DO statement starts a @DO procedure, i.e. the text lines and EDT statements in the specified work file are processed.
For information on the structure and processing of EDT procedures, see section “EDT procedures” on page 64.

Operation	Operands	F mode, L mode
@DO	procnr [,] [(param [,...])] [spec] [=line1,line2 [, [-] line3]] [**P**RINT]	

procnr
The number of the work file (`1..22`) whose content is to be processed by EDT.
If the work file is empty, error message `EDT4950` is issued.

param
Parameters which are passed to the procedure that is to be run. The parameters must be defined in the procedure using @PARAMS (see the @PARAMS statement). They are separated from one another by commas.

If parameters (including empty parameters) are specified in a procedure which contains no @PARAMS statement then the statement is rejected with the message `EDT4944`. If too many parameters are specified then the error message `EDT4963` or `EDT4965` is output.

A distinction is made between positional and keyword parameters. In the case of positional parameters, only the value of the parameter is passed. In the case of keyword parameters, an expression of the form `formal=value` is passed where `formal` is the keyword (not prefixed by the & character) with which the parameter was defined in the @PARAMS statement (detailed information on parameter transfer can be found in the description of the @PARAMS statement below).

The positional parameters must be located before the keyword parameters and must be specified in precisely the same order as they were defined in @PARAMS. Keyword parameters can be specified in any order. If a positional parameter is specified after a keyword parameter then the statement is rejected with `EDT4948`. If a keyword parameter is specified more than once then the message `EDT3911` is issued.

The possible number of parameters is limited by the maximum length of an EDT statement.

spec

Loop counter. In the procedure, this can be used as an operand in EDT statements if it is necessary to address a line number. When the procedure is executed, EDT uses the current value of the loop counter at all times (see section “EDT procedures” on page 64).

The loop counter must be one of the permitted special characters as otherwise @DO is rejected with the error message `EDT3952`. To avoid errors and unpredictable events, the following characters should not be used as a loop counter.

`% $ ? * ( : # + - . < = > ’ ;`

The command syntax means that the `'='` character cannot be specified at all. If the procedure is started in F mode then `';'` may not be used.

Suitable characters for the loop counter are:

`! { } [ ] | /`

If the loop counter is not specified then it is considered to be undefined. If the sequence of operands `line1,line2,[-]line3` is not specified then the loop counter has the value 1.

=line1,line2,[–]line3

A procedure is repeated several times (see example 3).

Before the first pass, EDT assigns the initial value `line1` to the loop counter. After every pass, EDT increments or decrements (minus sign in front of `line3`) the loop counter by `line3`. The default value for `line3` is 1. The procedure is repeated as long as the loop counter has not risen above (or fallen below) the value of `line2`. Otherwise the execution of the procedure is aborted.

The procedure is passed through at least once since the counter value is checked after the last line has been processed (`REPEAT UNTIL`). If the presence of a @RETURN statement in the procedure means that the last line is not processed then no check is performed and the procedure is not repeated.

Line number symbols (e.g. `%`, `$`) may also be specified for `line1`, `line2` or `line3`. EDT uses the value which this symbol had when @DO is executed. If the value of this symbol changes during execution of the procedure then the number of passes is not affected.

In the procedure, the loop counter is treated like a line number variable. The loop counter is therefore only replaced by the current value if it is addressed as a line number, in particular therefore *not* in literals. If the loop counter is

specified in a statement which permits a line number with an implicit increment then – in the same way as for line number variables - the implicit increment is always considered to be `0.0001` (see example 6.

The fact that a special character is only considered to be a loop counter if it is used instead of a line number and is specified as a loop counter in the calling @DO statement means that the same loop counter can be used more than once for nested external loops. In this case, it always has the values that are specified in the calling @DO statement (see example 7). It is not possible to access loop counters which have been specified in a lower nested level at call time. The symbols used there are not replaced by line numbers in higher levels and this usually results in syntax errors.

The default value for `line1`, `line2` and `line3` is 1.

PRINT — Each line of the procedure should be logged (with expanded parameters) before it is executed. In interactive mode, the output is written to `SYSOUT` and in batch mode it is written to `SYSLST`.

Specifying `PRINT` also causes all error messages to be output and sets the EDT error switch. Normally, the two messages `EDT0901` and `EDT4932` are not output in procedures and the EDT error switch is not set (see section “Message texts” on page 638).

The value of a positional parameter is determined by all the characters, including blanks, specified between the commas or parentheses.
If there are no characters between the commas or parentheses then the value of the positional parameter is a empty string.
The value of a keyword parameter is determined by all the characters, including blanks, specified between the equals sign and the following comma or parenthesis.
If there are no characters between the equals sign and the following comma or parenthesis then the value of the keyword parameter is a empty string.
A parameter may be enclosed in single quotes. These are not transferred if they occur as the first or last character in the parameter value and only double quotes occur between them.
In all other cases, the specified single quotes form part of the parameter (see the examples for the @PARAMS statement). The comma and closing parenthesis characters may only form part of a parameter if they occur in a substring in the parameter value that is enclosed by single quotes.
Single quotes must always occur in pairs in a parameter value. An individual single quote cannot be passed in a parameter value. If @QUOTE has already been used to assign the function of the single quote to another character then this does **not** apply to the single quotes enclosing the parameter value.
If a positional parameter is not specified then it is assigned the value of an empty string.
If a keyword parameter is not specified then it is assigned the default value defined in the @PARAMS statement.

During parameter substitution, the specified parameters are converted into the character set used by the procedure work file. If the string variable contains characters which cannot be displayed in the target character set then these are replaced by a substitute character if such a character has been specified (see @PAR SUBSTITUTION-CHARACTER). Otherwise, the @DO statement is rejected and the error message EDT5453 is output.

The parameters may contain Unicode substitute representations. These are not expanded during the parameter substitution process. This is not done until the associated procedure line is executed.

EDT procedures can be interrupted at any time using K2.
In the operating system, it is then possible to use /RESUME-PROGRAM to continue the procedure or use /INFORM-PROGRAM to return to EDT and abort the procedure.

The procedure cannot be aborted with /INFORM-PROGRAM while a user statement (see @USE) is being executed.

An illegal statement during execution does not cause the procedure to be aborted.

Example 1

```
1.     @SET #S0 = 'TEST OF PROCEDURE FILE 1'
1.     @PROC 1 ------------------------------------------------------- (1)
1.     @ @SET #S1 = #S0:1-4: ----------------------------------------- (2)
2.     @ @CREATE #S2: ' '*4,#S0:5-8:
3.     @ @CREATE #S3: ' '*9,#S0:9-24:
4.     @ @CREATE #S4: ' '*24
5.     @ @PRINT #S1.-#S4
6.     @ @PRINT #S0
7.     @END ---------------------------------------------------------- (3)
1.     @DO 1 --------------------------------------------------------- (4)
  #S01 TEST
  #S02      OF
  #S03          PROCEDURE FILE 1
  #S04
  #S00 TEST OF PROCEDURE FILE 1
1.
```

(1) Processing switches to work file 1.

(2) EDT statements are written to work file 1. When the procedure is called with @DO, these cause the string variables #S1 to #S4 to be created and output together with #S0.

(3) @END causes a return from work file 1.

(4) The procedure located in work file 1 is called.

Example 2

```
1.     @PROC 2 ------------------------------------------------------- (1)
1.     @ @PARAMS &STRING --------------------------------------------- (2)
2.     @ @SET #S1 = '++++++++++'
3.     @ @SET #S2 = &STRING ------------------------------------------ (3)
4.     @ @PRINT #S2
5.     @END
1.     @DO 2(#S1) PRINT ---------------------------------------------- (4)
1.     @SET #S1 = '++++++++++'
1.     @SET #S2 = #S1
1.     @PRINT #S2
  #S02 ++++++++++
1.     @DO 2('#S1') PRINT -------------------------------------------- (5)
1.     @SET #S1 = '++++++++++'
1.     @SET #S2 = #S1
1.     @PRINT #S2
  #S02 ++++++++++
1.     @DO 2(␣'#S1'␣) PRINT ------------------------------------------ (6)
1.     @SET #S1 = '++++++++++'
1.     @SET #S2 = ␣'#S1'␣
1.     @PRINT #S2
  #S02 #S1
1.
```

(1) Processing switches to work file 2.

(2) The first line stored in this work file is a @PARAMS statement. This makes it possible to address the positional parameter `&STRING` in this work file.

(3) `#S2` is to be assigned a value that is not available at the time work file 2 is defined and will only be defined in a @DO 2(...)... statement.

(4) The value `#S1` present in parentheses causes `&STRING` to be replaced globally by the value `#S1` before the statements located in work file 2 are executed. `PRINT` causes the statements to be output before they are executed.

(5) 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 removed when the parameter value is replaced in work file 2 as the `PRINT` operand here clearly shows. This therefore has the same effect as (4).

(6) The only difference to (5) is that the parameter value has been extended by a preceding or following blank. However, this is sufficient to ensure that the content of the parameter value is passed.

Example 3

```
  1.     *
  2.     @PROC 3 ------------------------------------------------------ (1)
  1.     @ @CREATE $+1: $,'*' ----------------------------------------- (2)
  2.     @END ---------------------------------------------------------- (3)
  2.     @DO 3,!=1,15 ------------------------------------------------- (4)
  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 **************** ------------------------------------------- (5)
  2.
```

(1) Processing switches to work file 3.

(2) A single EDT statement is written to work file 3.

(3) Processing returns to work file 0.

(4) Work file 3 is executed. In this case, the ! character is used as a loop counter. Work file 3 is executed 15 times. It would be possible to address line numbers there using !. However, this can be omitted as in this example. The specification `!=1,15` has the same effect as issuing @DO 3 fifteen times without this sequence of operands.

(5) In the output, it can be seen 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 ------------------------------------------------------- (1)
   1.     @ @PRINT !.-.$ ------------------------------------------------ (2)
   2.     @ @PRINT #S4 N
   3.     @END
   5.     @DO 4,!=$,%,-1 ------------------------------------------------ (3)
   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.
```

(1) Processing switches to work file 4.

(2) A line number is addressed via the loop counter `!`.

(3) Work file 4 is executed a number of times. On the first pass, the value of the highest assigned line number is assumed for `!`. On each subsequent pass, this value is reduced by 1 (third `line=-1`) until the line number has the value of the lowest assigned line number (`%`).

Example 5

```
1.     @PROC 4 ------------------------------------------------------- (1)
1.     @READ 'PROC-FILE.4' ------------------------------------------- (2)
6.     @PRINT
1.0000 @PARAMS &A, &OPTION=ALL
2.0000 ABCABCABCABC
3.0000 EFG
4.0000 @ON  1 CHANGE &OPTION 'ABC' TO '&A'
5.0000 @5: &A
6.     @END ---------------------------------------------------------- (3)
1.     @DO 4 ('A','B',OPTION=R) -------------------------------------- (4)
6.     @PRINT
1.0000 ABCABCABCA,'B ------------------------------------------------ (5)
2.0000 EFG
5.0000  A,''B
6.
```

(1) Processing switches to work file 4.

(2) The SAM file `PROC-FILE.4` is read into work file 4.

(3) Processing returns to work file 0.

(4) The default value `ALL` of the keyword parameter `&OPTION` is replaced by `R` at call time. As a result, the search and replace operation in line 1 is performed backwards.

(5) The execution of the work file caused lines to be written to the current work file. In line 1, EDT has removed one of the 2 successive single quotes. In line 3, EDT has taken over the parameter value unchanged.

Example 6

```
1.     AAA
2.     BBB
3.     CCC
4.     @PROC 1
1.     @@COPY 1-3 TO ! ------------------------------------------------ (1)
2.     @END
4.     @DO 1,!=4,4 ---------------------------------------------------- (2)
5.0002 @DO 1,!=5.0,5.0 ------------------------------------------------ (3)
6.0002 @PRINT --------------------------------------------------------- (4)
1.0000 AAA
2.0000 BBB
3.0000 CCC
4.0000 AAA
4.0001 BBB
4.0002 CCC
5.0000 AAA
5.0001 BBB
5.0002 CCC
6.0002
```

(1) The procedure file contains a statement in which the increment is implicitly defined by specifying the line number of the target range.

(2) Calls the procedure with loop counter `4`.

(3) Calls the procedure with loop counter `5.0`.

(4) The output shows that the implicit increment `0.0001` is always used for the loop counter as is also the case for line number variables.

Example 7

```
1.     @PROC 1
1.     @   @DO 2, !=!,1,-1 ------------------------------------------ (1)
1.     @END
1.     @PROC 2
1.     @   @SET #L1 = ! --------------------------------------------- (2)
2.     @   @SET #S1 = C #L1 ----------------------------------------- (3)
3.     @   @PRINT #S1 ----------------------------------------------- (4)
3.     @END
1.     @DO 1,!=1.4 -------------------------------------------------- (5)
  #S01    1.0000 ------------------------------------------------------ (6)
  #S01    2.0000
  #S01    1.0000
  #S01    3.0000
  #S01    2.0000
  #S01    1.0000
  #S01    4.0000
  #S01    3.0000
  #S01    2.0000
  #S01    1.0000
1.
```

(1) A @DO procedure is stored in work file 1. In this case, the loop counter `!` is significant for two reasons: on the one hand, it is redefined by the @DO procedure call in work file 2 and, on the other, the current value of the loop counter specified when work file 1 was called is taken over as the initial value. This is possible because substitution is only performed in locations where the `!` symbol is also used as a line number.

(2) In work file 2, the current value of the work file's loop counter is assigned to the line number variable `#L1`.

(3) The line number variable `#L1` is stored in printable form in `#S1`.

(4) The string variable `#S1` is output.

(5) When work file 1 is called, the loop counter is allowed to run from `1` to `4`.

(6) The result indicates that the internal loop counter (in work file 2) runs backwards 4 times until it reaches `1.0000` on the basis of the starting values `1.0000` to `4.0000`.

9.38 @DO (format 2) – Activate or deactivate logging

This format of the @DO statement can be used to suspend or activate the logging of the read statements (see the PRINT operand in format 1 of the @DO statement) at any location within the procedure.

Operation	Operands	@PROC
@DO	{ **N** / **P** }	

N EDT no longer logs the following lines of the procedure before they are executed.

P EDT logs the following lines of the procedure before they are executed.

This statement can also be used for troubleshooting in EDT procedures. It is possible to find out, for example, whether a specific part of a procedure has been processed or not.

Example

```
1.     @PROC 5 ------------------------------------------------------ (1)
1.     @ @SET #S5 = 'A'
2.     @ @DO N ------------------------------------------------------ (2)
3.     @ @CREATE #S6: 'B'*6,#S5
4.     @ @CREATE #S7: #S6,'C',#S6
5.     @ @DO P
6.     @ @PRINT #S5.-#S7 -------------------------------------------- (3)
7.     @ @DELETE #S5.-#S7
8.     @END --------------------------------------------------------- (4)
1.     @DO 5 PRINT -------------------------------------------------- (5)
1.     @SET #S5 = 'A'
1.     @DO N
1.     @PRINT #S5.-#S7
  #S05 A
  #S06 BBBBBBA
  #S07 BBBBBBACBBBBBBA
1.     @DELETE #S5.-#S7
1.
```

(1) Processing switches to work file 5.

(2) The lines that follow in the procedure are no longer logged.

(3) EDT logs the following lines of the procedure before execution.

(4) Processing returns to work file 0.

(5) The procedure in work file 5 is started. The statements are to be logged before being executed.

9.39 @DROP – Delete work files

The @DROP statement completely deletes the specified work files.

Operation	Operands	L mode
@DROP	{ procnr[,...] / **A**LL }	

procnr The number of the work file (1..22) that is to be deleted. Any number of work files can be specified.

ALL Work files (1..22) are deleted.

The @DROP statement may only be entered if the current work file is work file 0. @DROP is also not permitted in @DO procedures.

The work files are deleted without a confirmation query, irrespective of their content. If files have been opened with @OPEN or @XOPEN in the work files that are to be deleted then these are implicitly closed.

Note

The @DROP statement has the same effect as deleting each of the specified work files with @DELETE (format 2) and removes the specified work files from the set of work files in use (see @PROC statement).
Opened files or library elements should first be written back and closed (see @CLOSE) as otherwise any changes will be lost.

Example 1

```
   1.     @PROC USED ------------------------------------------------------ (1)
<03>     1.0000 TO    3.0000
<05>     1.0000 TO    1.0000
<08>     1.0000 TO    1.0000
<10>     1.0000 TO    1.0000
<14>     1.0000 TO    1.0000
   1.     @DROP 10 ------------------------------------------------------- (2)
   1.     @PROC USED
<03>     1.0000 TO    3.0000
<05>     1.0000 TO    1.0000 ---------------------------------------------- (3)
<08>     1.0000 TO    1.0000
<14>     1.0000 TO    1.0000
   1.     @DROP 8,5 ------------------------------------------------------ (4)
   1.     @PROC USED
<03>     1.0000 TO    3.0000 ---------------------------------------------- (5)
<14>     1.0000 TO    1.0000
   1.
```

(1) The work files `1..22` that are in use should be output. In this case, these are the work files `3`, `5`, `8`, `10`, `14`.

(2) Work file `10` is deleted and released.

(3) @PROC USED reports that only the work files `3`, `5`, `8`, `14` are still in use.

(4) @DROP can also be used to delete and release multiple work files such as, for example, `5` and `8` here.

(5) This only leaves work files `3` and `14`.

Example 2

```
   1.     @PROC USED ------------------------------------------------------ (1)
<03>     1.0000 TO    3.0000
<14>     1.0000 TO    1.0000
   1.     @DROP ALL ------------------------------------------------------ (2)
   1.     @PROC USED
% EDT0907 NO WORK FILES USED ------------------------------------------------ (3)
   1.
```

(1) All the work files that are in use should be output.

(2) The work files `1..22` are deleted and released.

(3) None of the work files `1` to `22` is now in use.

9.40 @EDIT (format 1) – Switch to F mode

In interactive mode, format 1 of the @EDIT statement switches from L mode to F mode.

Operation	Operands	F mode, L mode
@EDIT	**F**ULL [**SCREEN**]	

In batch mode and in F mode, this statement is ignored. If it occurs inside an EDT procedure, it is rejected with the message `EDT4920`.

If @EDIT FULL SCREEN is specified inside a statement block (`BLOCK` mode) in interactive mode then the statements that follow it are ignored.

Note

The statement @PAR EDIT-FULL differs semantically from @EDIT FULL SCREEN and cannot therefore be used as an alternative to @EDIT FULL SCREEN.

9.41 @EDIT (format 2) – Set input from terminal

In the interactive mode's L mode, format 2 of the @EDIT statement switches the input stream to terminal input. WRTRD is used for reading and the current line number is output as the prompt.

If the statement is entered in F mode, operation first switches to L mode. In batch mode, this only affects logging (see note below).

Operation	Operands	F mode, L mode
@EDIT	[**P**RINT] [**S**EQUENTIAL]	

PRINT Specifying PRINT causes the line number and content of the current line to be output on the screen before the prompt is output in interactive mode and before the next statement or line of data is read in batch mode.

If PRINT is not specified, the statement deactivates this function again without exiting L mode.

SEQUENTIAL The operand affects the incrementation of the current line number. Normally, in L mode the current line number is increased or decreased by the increment when a data line is entered in or when the statements @+ or @– are issued. This may result in existing lines, i.e. the lines between the old and the new current line number, being skipped without this being noticed by the user.

If SEQUENTIAL is specified then the current line number is only formed as described above if there are no intervening lines. If this is not the case, the first intervening line becomes the current line.

If SEQUENTIAL is not specified, the statement deactivates this function again without exiting L mode.

Note

In batch mode, EDT usually reads from SYSDTA. However, the type of logging that is performed (see @LOG statement) differs depending on whether @EDIT format 2 or @EDIT format 3 is specified. If @EDIT format 2 is specified then every logged entry starts with the current line number whereas this is omitted in @EDIT format 3. The latter behavior corresponds to the setting when EDT is started in batch mode.

9.42 @EDIT (format 3) – Set input from SYSDTA

In the interactive mode's L mode, format 3 of the @EDIT statement switches the input stream to input from SYSDTA. Reading is performed with RDATA. The type of prompt displayed and the method used depend on the operating system settings (by default, a * is output).

If the statement is entered in F mode, operation first switches to L mode. In batch mode, this only affects logging (see note below).

Operation	Operands	F mode, L mode
@EDIT	**O**NLY [**P**RINT] [**S**EQUENTIAL]	

PRINT Specifying PRINT causes the line number and content of the current line to be output on the screen before the prompt is output in interactive mode and before the next statement or line of data is read in batch mode.

If PRINT is not specified then only the input stream is switched and the SEQUENTIAL operand is evaluated if it has been specified.

If the function is activated with @EDIT ONLY PRINT then the only way to deactivate it without exiting L mode is to issue the @EDIT (format 2) statement.

SEQUENTIAL The operand affects the incrementation of the current line number. Normally, in L mode the current line number is increased or decreased by the increment when a data line is entered in or when the statements @+ or @– are issued. This may result in existing lines, i.e. the lines between the old and the new current line number, being skipped without this being noticed by the user.

If SEQUENTIAL is specified then the current line number is only formed as described above if there are no intervening lines. If this is not the case, the first intervening line number becomes the current line number.

If SEQUENTIAL is not specified then only the input stream is switched and the PRINT operand is evaluated if it has been specified.

If the function is activated with @EDIT ONLY SEQUENTIAL then the only way to deactivate it without exiting L mode is to issue the @EDIT (format 2) statement.

If the input is redirected to SYSDTA with @EDIT format 3 then statements and data are interpreted in the character set that is currently defined for SYSDTA. In interactive mode, this is usually the same as the character set declared for the terminal using /MODIFY-TERMINAL-OPTIONS unless SYSDTA was previously assigned to a file. If the statements and data refer to one of the EDT work files, it may be necessary to convert the input into the character set of the work file in question. For details, see section "Character sets" on page 47.

Note

In batch mode, EDT usually reads from SYSDTA. However, the type of logging that is performed (see @LOG statement) differs depending on whether @EDIT format 2 or @EDIT format 3 is specified. If @EDIT format 2 is specified then every logged entry starts with the current line number whereas this is omitted in @EDIT format 3. The latter behavior corresponds to the setting when EDT is started in batch mode.
In interactive mode, if an EOF is identified at SYSDTA, EDT automatically switches to terminal input.

9.43 @EDIT (format 4) – Control full record display

In F mode, format 4 of the @EDIT statement switches between the full display of records and the display of a record section in the current work file's data window.

Operation	Operands	F mode
@EDIT	**L**ONG [{ **ON** / **OFF** }]	

ON — The display in F mode is set in such a way that the records are (if possible) fully displayed in the data window. The line number display is deactivated. For details of the display in EDIT–LONG mode, see section "The work window" on page 103.

Activating EDIT–LONG mode implicitly deactivates the line number display (@PAR INDEX=OFF) and hexadecimal mode (@PAR HEX=OFF).

OFF — The display of long records in F mode is set in such a way that only a section (depending on the terminal and @VDT and @PAR INDEX setting, this may be 72, 80, 124 or 132 characters) is visible in the data window. For details on work file display, see section "The work window" on page 103.

The line number display remains active when EDIT–LONG mode is exited. EDIT–LONG mode is also deactivated by @PAR INDEX=ON and @PAR HEX=ON.

At the start of an EDT session, @EDIT LONG OFF is set by default for all the work files.

In EDIT–LONG mode, neither the column counter activated with @PAR SCALE=ON nor an information line requested with @PAR INFORMATION=ON are displayed. The column counter and information lines are not displayed until EDIT–LONG mode is exited.

The activation and deactivation of EDIT–LONG mode applies at work file level. If the relevant work file is displayed in multiple data windows on the screen then the same mode is used in both data windows.

The @PAR EDIT-LONG statement can be used instead of @EDIT format 4 and has the same functionality. Furthermore, @PAR EDIT-LONG can be used for a specific work file or globally for all the work files and is also permitted in L mode and therefore in EDT procedures.

9.44 @ELIM – Delete records in an ISAM file

The @ELIM statement deletes an ISAM file either fully or partially. If the entire content is deleted then - unlike @UNSAVE – the file name remains present in the catalog. It is also possible to delete the file in the work file and on disk simultaneously.

Operation	Operands	F mode, L mode
@ELIM	[file] [(ver)] lines[,...] [**B**OTH]	

file — Name of the ISAM file in which ranges are to be deleted. The name must correspond to the SDF data type `<filename 1..54>` or must consist of the special specification `'/'`.

If the `file` operand is not specified then the explicit local @FILE entry, if present, and otherwise the global @FILE entry is used as the file name (see also @FILE statement). If there is neither an explicit local nor a global @FILE entry then the @ELIM statement is rejected with the error message `EDT5484`.

If the specified file does not exist, is of the wrong type or cannot be accessed, then the @ELIM statement is rejected with a corresponding message.

If the file link name `EDTISAM` is assigned to a file then the user simply needs to specify `'/'` in order to delete records in the file (see chapter "File processing" on page 131).

ver — Version number of the file that is to be deleted. If the specified version number does not match the file's version number, the statement is rejected with message `EDT4985`.

lines — One or more line ranges that are to be deleted from the ISAM file. If symbolic line numbers are specified then their values are determined from the current work file and therefore usually have nothing to do with the record structure of the file specified in `file`.

If the range symbol (default value `&`) is specified for `lines` and the default value is still set to `0.0001-9999.9999` (see @RANGE statement) then the entire content of the file is deleted. However, the catalog entry is retained.

BOTH — The designated line range is to be deleted both in the ISAM file and in the work file.

The file is only opened temporarily during the delete operation. @ELIM can only be used to delete records in files that do not have the default attributes assumed by EDT (e.g. no variable record length) if a corresponding /SET-FILE-LINK command has been issued with the file link name EDTISAM (see chapter "File processing" on page 131).

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
  23.00 ·····························································
@get 'xmpl.elim' noreseq ·····································0001.00:00001(00)
```

The ISAM file XMPL.ELIM is read into work file 0. The ISAM keys are to be taken over 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<························································
@elim 'xmpl.elim' 5-7 both ····································0001.00:00001(00)
```

The line range 5-7 is deleted both in work file 0 and in the ISAM file XMPL.ELIM.
If the ISAM file is read using @GET without NORESEQ then the ranges to be deleted in the ISAM file and the work file do not have to correspond.

```
   1.00 ONE<··························································
   2.00 TWO<··························································
   3.00 THREE<························································
   4.00 FOUR<·························································
   8.00 EIGHT<························································
   9.00 ·····························································
@delete ; @get 'xmpl.elim' noreseq·····························0001.00:00001(00)
```

Work file 0 is deleted and then 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 ·····································································
```

The line range 5 to 7 is also deleted in the ISAM file.

9.45 @END – Exit current work file or terminate the EDT session

In L mode, @END causes the current work file to be exited. Processing returns to the work file in which the @PROC statement activating the current work file was issued. In F mode, @END terminates the EDT session or terminates the screen dialog.

Operation	Operands	F mode, L mode
@END	[comment]	

comment The `comment` operand may contain any text as a comment. It may only be input in L mode.

In this statement, it is obligatory for at least one blank to be entered between the statement name and any specified operands.

If the @END statement is entered in work file 0 in L mode then the message `EDT4939` is output. In batch mode and in EDT procedures, the next statement is then read. In interactive mode, behavior after the warning message `EDT4939` is the same as for the @HALT statement without operands except that even if job switch 4 is set, the queries `EDT0900` and `EDT0904` are still output.

In F mode, the @END statement is always processed in the same way as @HALT without operands independently of the current work file, i.e. the EDT session or screen dialog is terminated and, if EDT was called as a subroutine, control returns to the calling program (see @HALT statement).

If the current work file (not equal to work file 0) in L mode was not set by means of a @PROC statement but with @SETF or implicitly after a switch from F mode to L mode then processing switches back to work file 0.

@END cannot be used to make a work file in which a @DO procedure is running (active work file) the current work file. Any such attempt is rejected with the message `EDT4959`.

Example 1

```
   1.     @PROC 1 ------------------------------------------------------ (1)
   1.     @ @SET #S1 = DATE
   2.     @ @SET #S2 = TIME      ---------------------------------------- (2)
   3.     @ @PRINT #S1.-#S2 N
   4.     @END ---------------------------------------------------------- (3)
```

(1) Processing switches to work file 1.

(2) An EDT procedure is entered in work file 1.

(3) Processing returns to work file 0. The procedure located in work file 1 can be called with @DO 1.

Example 2

```
   1.     @PROC 7 ------------------------------------------------------ (1)
   1.     @PROC -------------------------------------------------------- (2)
<07>
   1.     @ @SET #S7 = 'THIS IS PROC 7'
   2.     @ @PRINT #S7
   3.     @PROC 8 ------------------------------------------------------ (3)
   1.     @ @SET #S8 = 'THIS IS PROC 8'
   2.     @PROC USED
<07>     1.0000 TO    2.0000   ---------------------------------------- (4)
<08>     1.0000 TO    1.0000
   2.     @END --------------------------------------------------------- (5)
   3.     @PROC -------------------------------------------------------- (6)
<07>
   3.     @END --------------------------------------------------------- (7)
   1.     @PROC -------------------------------------------------------- (8)
<00>
```

(1) Processing switches to work file 7.

(2) Queries the current work file.

(3) Processing switches to work file 8.

(4) Work files 7 and 8 are in use.

(5) EDT returns to work file 7 (from where processing branched to work file 8 due to @PROC).

(6) The query of the current work file confirms the return to work file 7.

(7) Processing returns to work file 0 again.

(8) Repeated query of the active work file confirms the return to work file 0.

9.46 @ERAJV – Delete job variables

The @ERAJV statement deletes job variable entries from the catalog.

Operation	Operands	F mode, L mode
@ERAJV	string [**A**LL]	

string — String indicating the name of the job variable that is to be deleted. All the specifications that are also permitted in the BS2000 macro ERAJV are allowed. It is therefore also possible to make a partially qualified entry or use wildcards. EDT does not perform any full syntax check.

If the specification does not designate any existing job variable then message EDT4982 is output.

ALL — If ALL is specified then all the job variables designated by string are removed from the catalog without any confirmation query.

If ALL is not specified and more than one job variable indicated by string is present then the statement is not executed in batch mode. In interactive mode, EDT issues the query

```
%  EDT0298 ERASE ALL JOBVARIABLES (&00)? REPLY (Y=YES; N=NO)?
```

If the user responds N then the message EDT0299 is output and the statement is aborted.

If the BS2000 macro ERAJV is rejected by the system (for example, if the job variable is password-protected) then EDT reports the error EDT4208.

If the Job Variable Support subsystem is not installed, the statement is rejected with the error message EDT5254. For details concerning job variables, see the User Guide JV [9].

9.47 @EXEC – Start program

The @EXEC statement terminates the EDT session and loads and starts the specified program.

Operation	Operands	F mode, L mode
@EXEC	string	

string String specifying the name of the program that is to be loaded and started. The system expects the name of a BS2000 file which contains the program that is to be loaded. It is not possible to specify a library element.

The @EXEC statement is one of the EDT statements with security implications (see also section "Access protection" on page 99). The statement is rejected in uninterruptible system procedures in interactive mode and on input from a file (read with RDATA from SYSDTA not equal to SYSCMD, execution of a start procedure).

The @EXEC statement always causes EDT to be terminated irrespective of whether the specified program file exists or contains a valid program.

As far as the handling of unsaved files and the related security queries is concerned, @EXEC acts in the same way as the @HALT statement (see section "Terminating an EDT session" on page 92). Since EDT is always terminated, save queries may, unlike in the case of @HALT, also be issued if the statement was entered in the screen dialog (started with @DIALOG).

If EDT was loaded as a subroutine and the EDT screen dialog has been activated with @DIALOG, the @EXEC statement does not result in the continuation of the subroutine. Instead, the user program is also unloaded.

It is therefore possible to prohibit users from issuing the @EXEC statement when EDT is called as a subroutine. In this case, calls are rejected with message EDT4976.

Note

If @DIALOG was entered in a system procedure, then the remaining procedure commands after @EXEC may be interpreted as input for the newly started program and may therefore result in unexpected effects.

Example

The example assumes that the records present in the work file have not yet been saved.

```
   1.00 EDT is to be terminated<·····································
   2.00 and LMS is to be loaded<·····································
   3.00 This is done with the @EXEC statement<·····················
   4.00 ·····························································
@exec '$lms'·····················································0001.00:00001(00)
```

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 yet been saved, EDT queries (in the same way as in @HALT), whether it should really terminate. Only if the user responds `Y` is EDT terminated and LMS loaded and started.

9.48 @FILE – Preset file name

The @FILE statement can be used to preset a file name for @GET, @READ, @WRITE, @SAVE, @OPEN (format 2) and @ELIM. It is also possible to predefine a file name that only applies to the current work file (explicit local @FILE entry), or a file name which applies to all the work files (global @FILE entry).

Operation	Operands	F mode, L mode
@FILE	[string [(ver)]] [**LOC**AL]	

string — String specifying a file name. The name must correspond to the SDF data type `<filename 1..54>` or must consist of the special specification `'/'`.

ver — Version number of the file. If `LOCAL` is specified then the specification of the version number has no effect. The version number can be overwritten by means of an explicit specification in the file access statements.

LOCAL — The specified file name is recorded as the work file-specific file name of the current work file (explicit local @FILE entry). If string is not specified then the local @FILE entry is deleted.

If `LOCAL` is not specified then the specified file name is recorded as the global @FILE entry. If `string` is also not specified then the global @FILE entry is deleted.

If there is no explicit local @FILE entry when the statements @READ 'file' or @GET 'file' are executed then the specified file name becomes the local file name (implicit local @FILE entry).

In the case of the @WRITE and @SAVE statements, the file name is searched for first in the statement, then in the explicit local @FILE entry, then in the global @FILE entry and finally in the implicit local @FILE entry.

In the case of the @GET, @READ and @ELIM statements, the file name is searched for first in the statement, then in the explicit local @FILE entry and finally in the global @FILE entry. These statements ignore an implicit local @FILE entry.

In the @OPEN statement (format 2), the file name is first searched for in the statement and then in the global @FILE entry. The @OPEN statement ignores local @FILE entries.

Note

The local @FILE entry is also deleted if the work file is completely deleted with @DELETE (format 2) or @DROP or if a file opened for real processing is closed with @CLOSE.

Example

```
  23.00 ·····························································
file 'xmpl.file' (*) ; get ·····································0000.00:00001(00)
```

The file name XMPL.FILE with the version number * is preset for the following @GET and @SAVE statements. The file XMPL.FILE is then read in with @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 ·····························································0001.00:00001(00)
```

The line range 1 to 2 is deleted in the work file and the content of the work file is then written to the file XMPL.FILE with @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) ·····························0003.00:00001(00)
```

```
   3.00 THREE<···························································
   4.00 FOUR<····························································
   5.00 FIVE<····························································
   6.00 ·····························································




%  EDT0902 FILE 'XMPL.FILE' VERSION 003
·····························································0003.00:00001(00)
```

9.49 @FSTAT – Output BS2000 catalog information

The @FSTAT statement can be used to output a list of files from the BS2000 catalog. It is possible to define the destination for the output. Optionally, it is also possible to output additional information about the files. The list is alphabetically sorted on the file names.

Operation	Operands	F mode, L mode
@FSTAT	[{file / svarex}] [[**T**O] line [(inc)]] [{**S**HORT / **LONG** [ISO**4**]}]	

file — Designates the files that are to be listed. The `file` operand must correspond to the SDF data type `<partial-filename 1..54 with-wild(80)>`.

Here, the symbolic name `'/'` for a file for which the `LINK` name `EDTSAM` or `EDTISAM` has been assigned by means of the `SET-FILE-LINK` command is not permitted.

If no file with the specified name is found, the message `EDT5281` is output.

svarex — The list of files for output can also be specified by means of a string variable (`#S00..#S20`).

line — Line number as of which information is to be written to the current work file. Any existing lines are overwritten.

If a line with a number greater than the previous highest line number is created then the current line number is modified.

If `line` is not specified then in the interactive mode's L mode, the result is output to `SYSOUT`, in batch mode it is output to `SYSLST` and in F mode it is written to work file `9`. Work file `9` is deleted before being used. If a file is open in work file `9` then the message `EDT5189` is output and the statement is not executed.

inc — Increment used to form the line numbers which follow `line`. If `inc` is not specified then the increment implicitly specified by `line` is used (see section “Implicit increment assignment” on page 35).
This specification does not change the work file's current increment.

SHORT — One file is output per line. Only file names, together with the associated catalog ID and user ID, are output.

LONG Further catalog information is output in addition to the file names.

Column	Header	Meaning
1–7	SIZE	Number of PAM pages
8	P	File on private or public data medium (*/␣)
9–62	FILENAME	File name with CATID and USERID
63–69	LAST PP	Last used PAM page
71–78	CR-DATE	Creation date (format YY-MM-DD)
80	S	SHARE attribute (Y/N/S)
81	A	ACCESS attribute (W/R)
83–86	FCB	FCB type (SAM/ISAM/PAM/BTAM/NONE)
88	R	READ-PASS attribute (Y/N)
89	W	WRITE-PASS attribute (Y/N)
91–98	CODESET	Character set

The list is alphabetically sorted on the file names.

If the output is sent to SYSOUT or SYSLST then it is accompanied by a header. If the output is sent to work file 9 (in F mode, without the line operand) then the header is output in the information line (can be displayed using @PAR INFORMATION=ON). If the output is written to a work file due to the line operand then no header is output.

ISO4 The creation date is output in the form YYYY-MM-DD. The following fields (see table) are moved accordingly.

If neither file nor svarex is specified then a list of all the files under the user's own ID is output.

If neither SHORT nor LONG is specified then only the file names are output (one per line). If the file specification in file or svarex contains a catalog ID, the file names are output with the catalog ID and user ID.

Otherwise, the file names are output in the same form as they are specified in the statement.
Partially qualified file names with a user ID form an exception here. In this case, @FSTAT outputs a list of file names without catalog IDs or user IDs for reasons of compatibility.

Output to SYSOUT or SYSLST is sent in the character set that has been defined for these system files. If the output is written to a work file then it is sent in the work file's character set. If the work file is empty and has the character set *NONE then the character set EDF041 is used. Characters that cannot be displayed in the target character set are always replaced by blanks.

Caution

In the case of large files, the fields for the reserved and used size for output in LONG mode may not be sufficient. In this case, nothing is output here. To ensure complete output in these cases, only the @SHOW command (format 1) should be used.

9.50 @GET – Read ISAM file

The @GET statement fully or partially reads an ISAM file from disk into the current work file.

Operation	Operands	F mode, L mode
@GET	[file] [(ver)] [lines[,...]] [:cols[,...]:] [**N**ORESEQ]	

file — Name of the ISAM file that is to be read in. The name must correspond to the SDF data type <filename 1..54> or must consist of the special specification '/'.

If there is as yet no local @FILE entry for the work file then, if the statement is successful, the specified file name is entered as an implicit local @FILE entry. If the file operand is not specified then, if present, the explicit local @FILE entry and otherwise the global @FILE entry is used as the file name (see also @FILE statement). If neither an explicit local nor a global @FILE entry is defined (e.g. there is only an implicit local file entry) then the @GET statement is rejected with the error message EDT5484.

If the specified file does not exist or cannot be accessed as required then the statement is rejected with a corresponding error message.

If the file link name EDTISAM is assigned to a file then the user simply needs to specify '/' in order to read this file (see chapter "File processing" on page 131).

ver — Version number of the file that is to be read. If the specified version number does not match the file's version number, the message EDT0902 is output and the file is read in nevertheless.

lines — One or more line ranges that are to be read in from the ISAM file. If symbolic line numbers are specified then their values are determined from the current work file and therefore usually have nothing to do with the record structure of the file specified in file.

If lines is not specified, the entire file is read in.

The line numbers specified with lines always refer to the record keys even if these are not taken over as line numbers. Consequently, if lines is specified, a check is always performed for valid record keys (see NORESEQ).

cols
: One or more column ranges which define the section to be read in from each record. The ranges may repeat and overlap. The column specifications refer to the characters in the file that is to be read in.
In the case of files which are present in a Unicode character set, they do not usually correspond to the byte positions within a record. If column values which exceed the record length are specified then blanks are read into the work file in their place.
Columns are counted starting after the record key.

  If no column range is specified then the lines are read in full.

NORESEQ
: The line numbers are formed from the ISAM keys of the ISAM files that are read. When this is done, lines which have existing line numbers may be overwritten.

  In this case, EDT checks whether a valid line number is present in the record key. To be valid, the key may consist only of the digits `0` to `9`. Otherwise, the @GET statement is aborted with the error message `EDT4984`. The records read up to this point are taken over into the work file. If a record has the key 0 then it is treated in the same way as a record with 1 (line number `0.0001`) and the warning `EDT2900` is issued.

  If `NORESEQ` is not specified then the line numbers are assigned as a function of the current line number and current increment (see section "Line number assignment" on page 36).

The file is only opened during the read operation. Records which consist solely of the key are read in as empty lines. If the file to be read in is empty, warning `EDT2903` is output.

If the current work file is empty and has the character set `*NONE` then it is assigned the character set of the file that is to be read in. If this character set is `*NONE` then the work file is assigned the character set `EDF03IRV`.

If the work file already has a character set then the records that are to be read in are converted from the file's character set into the work file's character set. If the file that is to be read contains characters which cannot be displayed in the work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise file read is aborted and error message `EDT5453` is output. This also applies if there are invalid characters outside of the column range that is to be read. In contrast, invalid characters outside of the line range that is to be read in are ignored.

If the file is present in a Unicode character set and contains an illegal byte sequence, e.g. surrogate characters, then it will be impossible to read it even if `SUBSTITUTION-CHARACTERS` is specified. In this case, the read operation is rejected with the message `EDT5454`.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Note

If a attempt is made to use @GET to read a SAM file then EDT issues the error message EDT1902 and sets the switch for EDT errors. It nevertheless reads the specified file by performing an internal @READ for this file. In this case, the lines or NORESEQ operands are ignored.

9.51 @GETJV – Read value of job variable

The @GETJV statement outputs the value of a job variable on the screen, writes it to a work file or assigns it to a string variable.

Operation	Operands	F mode, L mode
@GETJV	[string] [=⟨ line / svarex ⟩] [,**C**ODE=name]	

string — String which specifies the fully qualified name of a job variable. Although the name must comply with the syntactic rules for job variable names, EDT does not check these rules in full. If `string` is not specified then the job variable is addressed using the file link name `*EDTLINK`. If this is not defined then the message `EDT5289` is output.

line — Number of the line to which the value of the job variable is to be written.

svarex — String variable in which the value of the job variable is to be written.

name — Name of the character set in which the value of the job variable is to be interpreted. The character set must be valid; otherwise, the statement is rejected with message `EDT4980`. If the operand is not specified, the value of the job variable is interpreted in the character set `EDF041` (see section "Character sets" on page 47).

If neither `line` nor `svarex` is specified, the value of the job variable is output to `SYSOUT` in interactive mode and to `SYSLST` in batch mode.

If a line with a number greater than the previous highest line number is created then the current line number is modified.

If the job variable does not exist then the message `EDT4982` is output. If it cannot be accessed then the message `EDT4208` is output.

If the job variable is empty then an empty string is used as its value.

If the job variable contains an invalid byte sequence (possible in Unicode character sets) then it is not read and the message `EDT5454` is output.

If it is assigned to a string variable then this is also assigned the character set specified implicitly or explicitly via `name`. If it is inserted in a work file then the value is converted into the work file's character set. If the work file is empty and has the character set `*NONE` then it is assigned the character set specified explicitly or implicitly via `name`. If the string that is to be assigned contains characters which cannot be converted into the work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the string is not assigned and the error message `EDT5453` is output.

If the Job Variable Support subsystem is not installed, the statement is rejected with the error message `EDT5254`. For details concerning job variables, see the User Guide JV [9].

9.52 @GETLIST – Read elements of a list variable

The @GETLIST statement writes elements of a list variable into the current work file.

Operation	Operands	F mode, L mode
@GETLIST	string [lines[,...]] [:cols[,...]:] [,**C**ODE=name]	

string — String which specifies the name of an S list variable.
Although the name must comply with the syntactic rules for S variable names, EDT does not check these rules in full. If the name is longer than 246 characters then the statement is aborted with the message EDT3174.

lines — One or more line ranges which specify the list elements which are to be taken over.

Only the list variable elements that are identified by this line range are read. Element names and line numbers are assigned to one another in such a way that 0.0001 stands for the 1st element in the list, 0.0002 for the 2nd element etc.

If no element corresponds to a specified line number then the specification is ignored.

If lines is not specified, then all the elements are read.

cols — One or more column ranges in the S variables that are to be read. The ranges may repeat and overlap. Column n is assigned to the nth character. All the specified characters are concatenated in the sequence in which the columns are specified (possibly multiple times) and the result is inserted in the work file. If the result is longer than 32768 characters then the statement is aborted with the message EDT5474.

If a list element contains fewer characters than the specified column then a blank is inserted for it.

If no column range is specified then each element is read in full.

name — Name of the character set in which the value of the S variable is to be interpreted. The character set name must be permitted; otherwise, the statement is rejected with message EDT4980. If the operand is not specified, the value of the S variable is interpreted in the character set EDF041 (see section "Character sets" on page 47).

If the specified S variable is not a list then the message EDT4910 is output. If the value of a list element is not of type STRING then the statement is aborted with the message EDT5343. If the list does not contain any elements then the message EDT5340 is output.

Line numbers are assigned using the procedure “Insertion at the current line number” (see section “Line number assignment” on page 36). If the maximum line number is reached, the statement is aborted and the message EDT5252 is output.

If a list element contains an invalid byte sequence (possible in Unicode character sets) then the statement is aborted and the message EDT5454 is output.

On insertion, the values are converted into the work file's character set. If the work file is empty and has the character set *NONE then it is assigned the character set specified explicitly or implicitly via name. If the string that is to be assigned contains characters which cannot be converted into the work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the string is not assigned and the error message EDT5453 is output.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

For details on S variables, see the SDF-P User Guide [7].

9.53 @GETVAR – Read S variable

The @GETVAR statement outputs the value of an S variable, writes it to a work file or assigns it to a variable.

Operation	Operands	F mode, L mode
@GETVAR	{ string [={line / svarex / ivar}] / **SYSEDT** } [,**C**ODE=name]	

string — String which specifies a valid S variable name. Although the name must comply with the syntactic rules for S variable names, EDT does not check these rules in full.

line — Number of the line to which the value of the S variable is to be written.

If the value of the S variable is not of type `STRING` then the statement is aborted with the message `EDT5342`.

svarex — String variable in which the value of the S variable is to be written.

If the value of the S variable is not of type `STRING` then the statement is aborted with the message `EDT5342`.

ivar — Integer variable (`#I0..#I20`) into which the content of the S variable is to be taken over.

If the value of the S variable is not of type `INTEGER` then the statement is aborted with the message `EDT5342`.

SYSEDT — If they exist and their values are of the type `STRING` then the contents of the S variables `SYSEDT-S00..SYSEDT-S20` are assigned to the string variables `#S00..#S20`. In the case of non-existent S variables, S variables with no value or S variables of a different type, no error is reported. Instead the associated string variable is not modified.

name — Name of the character set in which the value of the S variable is to be interpreted. If the value of the S variable is not of type `STRING` then the specification is ignored. The character set name must be permitted; otherwise, the statement is rejected with message `EDT4980`. If the operand is not specified, the value of the S variable is interpreted in the character set `EDF041` (see section “Character sets” on page 47).

If neither line nor svarex nor ivar is specified then the value of the S variable is output to SYSOUT in interactive mode and in batch mode it is output to SYSLST.

If the S variable is not present then the message EDT5274 is output. If it has no value then the message EDT5340 is output (except in the case of SYSEDT).

If a line with a number greater than the previous highest line number is created then the current line number is modified.

If the variable contains an invalid byte sequence (possible in Unicode character sets) then it is not read and the message EDT5454 is output.

If it is assigned to a string variable then this is also assigned the character set specified implicitly or explicitly via name.
If it is inserted in a work file then the value is converted into the work file's character set. If the work file is empty and has the character set *NONE then it is assigned the character set specified explicitly or implicitly via name.
If the string that is to be assigned contains characters which cannot be converted into the work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the string is not assigned and the error message EDT5453 is output.

For details on S variables, see the SDF User Guide [6].

9.54 @GOTO – Branch statement in procedures

The @GOTO statement is used in a @DO procedure to execute an unconditional branch to the specified line.

Operation	Operands	@PROC
@GOTO	line	

line The `line` operand designates the line number to be branched to.

The @GOTO statement is only permitted in @DO procedures. If @GOTO is specified outside of a @DO procedure then it is rejected with the message `EDT4942`. In an @INPUT procedure, the message is output and processing continues with the statement which follows the illegal @GOTO statement.

If `line` is a line number variable which has the value `0.0000` at the time the branch is executed (this corresponds to the preset values of line number variables when EDT is started), then the @GOTO statement is rejected with the message `EDT4932` and the procedure continues with the statement which follows the invalid @GOTO statement.

The line which is branched to with @GOTO must exist in the associated procedure. If an attempt is made to branch to a line which does not exist then error message `EDT4974` is issued and the procedure continues with the statement which follows the invalid @GOTO statement.

If lines are to be branched to by means of @GOTO in EDT procedures then it is advisable to always define the line numbers of these lines explicitly by means of the @SET statement (format 6) in order to ensure that the numbers of the lines that are to be branched to do not change implicitly if statements are inserted or deleted.

Note

It is not advisable to use symbolic line numbers since these always refer to the current work file and not therefore to the procedure work file.
If the first statement in a procedure is the @PARAMS statement then this cannot be branched to with @GOTO.

Example

```
   1.     @SET #I3 = 1 ------------------------------------------------- (1)
   1.     @PROC 3
   1.     @1 ----------------------------------------------------------- (2)
   1.     @ @IF #I3 > 5 : @RETURN -------------------------------------- (3)
   2.     @ @STATUS = #I3
   3.     @ @SET #I3 = #I3+1
   4.     @ @GOTO 1
   5.     @END
   1.     @DO 3 -------------------------------------------------------- (4)
#I03= 0000000001
#I03= 0000000002
#I03= 0000000003
#I03= 0000000004
#I03= 0000000005
   1.
```

(1) The value 1 is assigned to the integer variable #I3.

(2) Specification of the line number which is branched to by means of @GOTO.

(3) If the procedure is executed in work file 3 then the value assigned to the integer variable #I3 should be incremented by 1 and output until it is greater than 5. This is implemented by means of a loop. At the end of the loop, @GOTO branches back to the start of the loop.

(4) The procedure in work file 3 is executed.

9.55 @HALT – Terminate EDT

The @HALT statement terminates the EDT session, the screen dialog after @DIALOG or EDT if it has been called as a subroutine with or without transferring a text to the calling program (see section “Terminating an EDT session” on page 92 for the general consequences of terminating EDT).

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 continues at the next JOB-STEP or in an ERROR-BLOCK.

If EDT was called as a subroutine then the entire string as of the first non-blank character after @HALT is passed to the calling program as a message text (see also message operand). If this string starts with 'ABNORMAL' then a special return code (0008002C instead of 00080000) is set.

message String which is passed to the calling program when EDT is called as a subroutine. This operand may only be specified if EDT is called as a subroutine.

If this string starts with 'ABNORMAL' then a special return code (0008002C instead of 00080000) is set.

In this statement, it is obligatory for at least one blank to be entered between the statement name and any specified operands.

The @HALT statement causes the termination of the EDT session in interactive or batch mode, if EDT was started as a main program with the /START-EDT or /START-EDTU command or equivalent /START-PROGRAM command (see section “Starting EDT” on page 87) and is not in screen dialog mode after @DIALOG. The EDT session is also terminated if @HALT is entered via the subroutine interface's CMD function (see the Subroutine Interfaces User Guide [1]).

In interactive mode, if the screen dialog is started with the @DIALOG statement from within a system procedure or via the subroutine interface then @HALT simply terminates the screen dialog and the system procedure or calling program is continued. The content of the work files is retained and there is consequently no save confirmation query (see below). In this case, specifying ABNORMAL has no special effect apart from informing the calling program.

If there are still any unsaved work files then, in interactive mode, the numbers of the work files with unsaved data are output after the message EDT0900 before EDT is terminated.

In addition, if they exist, the local @FILE entry (see the statements @FILE, @READ, @GET and @OPEN, format 2) or the name of an open file or the library and element name of an open library element (see the statements @OPEN and @XOPEN) is output for each of these work files.

In interactive mode, the user then sees the query:

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

If the user responds N then, in F mode, the work window is displayed. In L mode, the prompt is displayed again. The user may close or write back unsaved work files. If the user replies Y then unsaved work files are lost. EDT is terminated.

If job switch 4 is set before EDT is called, the save query is not issued. The save query is also not output if F mode has been called with @DIALOG (see also section "Terminating an EDT session" on page 92).

Example

```
1.     @HALT
%  EDT0900 EDITED FILE(S) NOT SAVED!
  LOCAL FILE ( 0) :
  LOCAL FILE ( 1) :
  LOCAL FILE ( 4) : L= EDT164
                    E= HALT(001),X
%  EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)?
```

@HALT terminates EDT in L mode. Since unsaved files are still present, the message EDT0900 is output together with a list of the work files.

9.56 @HEX – Set hexadecimal mode

The @HEX statement activates or deactivates hexadecimal mode for the current work file. In hexadecimal mode, all the records are displayed on screen in both printable and hexadecimal form.

Operation	Operands	F mode
@HEX	[{ **ON** | **O**FF }]	

ON Activates hexadecimal mode (default value).
This implicitly deactivates EDIT-LONG mode.

OFF Deactivates hexadecimal mode.

The layout in hexadecimal mode is described in detail in section “F mode” on page 101.

When an EDT session starts, hexadecimal mode is deactivated for all the work files.

The activation and deactivation of hexadecimal mode applies at work file level. If the relevant work file is displayed in multiple data windows on the screen then the same mode is used in both data windows.

If, in the case of split screen display, the work window is so small that it is not possible to display even one data line together with its hex lines then the message EDT2404 is output. Hexadecimal mode is activated nevertheless. The user can then enlarge the data window so that the hex lines can also be displayed.

The @PAR HEX statement can be used instead of @HEX and has the same functionality. Furthermore, @PAR HEX can be used for a specific work file or globally for all the work files and is also permitted in L mode and therefore in EDT procedures.

9.57 @IF (format 1) – Query error switches

This format of the @IF statement can be used in EDT procedures and in L mode to check whether EDT or DMS errors have already occurred. Depending on the result, a specified string either is or is not processed as input.

EDT errors may occur, for example, if an incorrect EDT statement is entered. The DMS error switch can be set for statements which access files (e.g. @WRITE, format 1) or may indicate system access errors.

Operation	Operands	L mode
@IF	{ **E**RRORS **N**O [**E**RRORS] **D**MS [**E**RRORS] **N**O **D**MS [**E**RRORS] } :[text]	

ERRORS The condition is fulfilled if the EDT error switch is set.

NO ERRORS The condition is fulfilled if the EDT error switch is *not* set.

DMS ERRORS
The condition is fulfilled if the DMS error switch is set.

NO DMS ERRORS
The condition is fulfilled if the DMS error switch is *not* set.

text EDT statement or data line. If the condition is fulfilled, the string is treated as if it had been entered at the prompt in L mode. In particular, the decision to interpret the text as data input or as a statement is made in accordance with the same rules (for more information, see section “L mode” on page 126).

The `text` operand starts immediately after the character `':'`, i.e. any specified blanks form part of the operand and are taken over into the line in the case of data input.

If `text` is not specified (although the colon is), then an empty line (line of length 0) is inserted.

Note

If a specific statement is to be checked then the error switch must be reset before the relevant statement (see @RESET). Otherwise the @IF statement may return an unwanted result since earlier statements may have already set the EDT or DMS switch.

The @IF ERRORS statement should not be used to query hits after @ON. @IF format 3 should be used for this.

Using @IF with @RETURN as a statement outside of procedures may cause EDT to terminate (see the @RETURN statement).

9.58 @IF (format 2) – Compare strings, line numbers and numbers

This format of the @IF statement can be used in EDT procedures to compare strings, line numbers or integer numbers with one another. The operands may be specified either explicitly (as literals) or via a reference (line number or EDT variable).

If the condition for the comparison is fulfilled then the specified string is processed as input. If the condition is not fulfilled, the statement has no effect.

Operation	Operands	L mode
@IF	{ [**S**] string1 rel string2 / line1 rel line2 / [**I**] int1 rel int2 } :[text]	

S — The keyword `S` is only obligatory if `string1` and `string2` contain line numbers or line number variables without column specifications, in which case it specifies that the *content* of the lines identified by the line numbers or knowledge variables is to be compared and not the line numbers themselves.

For example, the statement @IF #L1=#L2 compares the line numbers in `#L1` and `#L2`. However, if the contents of the lines indicated by `#L1` and `#L2` are to be compared then @IF S #L1=#L2 must be entered.

string1, string2 — The strings to be compared.

line1, line2 — The line numbers to be compared. The contents of the lines are not compared.

I — The keyword `I` only has to be entered if a number (a literal) has been explicitly entered for `int1` as otherwise EDT cannot tell whether the input is a line number or an integer.

int1, int2 — The integers to be compared.

A (positive or negative) integer or an integer variable can be entered for each of these (`#I0..#I20`).

rel Defines the relational operator:

Symbol			Meaning
>	or	GT	greater than
<	or	LT	less than
>=	or	GE	greater than or equal to
<=	or	LE	less than or equal to
=	or	EQ	equal to
<>	or	NE	not equal to

text EDT statement or data line. If the condition is fulfilled, the string is treated as if it had been entered at the prompt in L mode. In particular, the decision to interpret the text as data input or as a statement is made in accordance with the same rules (for more information, see section "L mode" on page 126).

The `text` operand starts immediately after the character `':'`, i.e. any specified blanks form part of the operand and are taken over into the line in the case of data input.

If `text` is not specified (although the colon is), then an empty line (line of length 0) is inserted.

The previous specification of `GOTO` or `RETURN` without a colon in procedures continues to be supported for reasons of compatibility.

Comparisons of two strings depend, on the one hand, on the character set in which the strings are encoded and, on the other, on the length of the strings (strings of zero length are permitted).

In all cases, it is possible to assign the operands `string1` and `string2` a character set which corresponds to their source. If the two character sets determined in this way are identical then a binary comparison is performed in this shared character set.
If the two character sets are different but are both EBCDIC 7/8-byte character sets then a binary comparison is performed as in the past.
Otherwise, the two character sets are converted internally into `UTF16` and the comparison is performed in `UTF16`.

Following any necessary conversion, the corresponding characters in the two strings are compared. Processing thus either reaches a non-identical character pair or the end of one or other of the two character strings. If the characters differ at any point then the two strings

are considered to be non-identical. EDT interprets the two non-identical characters as binary numbers on the basis of their character sets. The string with the character with the larger binary number is considered to be the greater of the two. If no non-identical pair of characters is identified then the longer of the two strings is considered to be the greater. If the two strings are of the same length and no non-identical character pair is detected then the strings are identical. If the two strings are of different lengths they can therefore never be identical.

Note

When line numbers are compared, both`%+3L` and `%+3` designate valid line numbers. This can lead to difficulties of interpretation if followed by the relational operator `LE`. It is therefore usually advisable to use the mathematical symbols for the relational operators.
The use of column specifications with the `string2` operand can also result in difficulties of interpretation when the `text` operand is introduced. Special care must be taken to ensure that the notation can be interpreted unambiguously in such cases. In particular, it is obligatory to conclude the column specification with a colon (otherwise optional).

For reasons of compatibility, the *sort weighting* defined by XHCS when comparing characters is ignored. This also applies to Unicode character sets. A sort operation, for example using the `SORT` program, may therefore return a sequence different from the result supplied by an @IF query in EDT.

Using @IF with @RETURN as a statement outside of procedures may cause EDT to terminate (see the @RETURN statement).

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' ---------------- (1)
   4.     @SET #I9 = 2
   4.     @PROC 3
   1.     @ @IF S %+#I9 = $-2L : @GOTO 4 ------------------------------- (2)
   2.     @ @PRINT #S1 N
   3.     @ @RETURN
   4.     @ @PRINT #S0 N
   5.     @END
   4.     @DO 3 -------------------------------------------------------- (3)
FIRST LINE = LAST LINE
   4.     @ON 1 DELETE 'NOT'
   4.     @DO 3 -------------------------------------------------------- (4)
FIRST LINE NOT EQUAL TO LAST LINE
   4.
```

(1) The string variables #S0 and #S1 as well as the integer variable #I9 are filled with content.

(2) When work file 3 is executed, line contents are compared here instead of line numbers.

(3) The execution of the statements stored in work file 3 results in the comparison of lines (%+#I9) and 1 ($-2L, i.e. 3-2). Since the contents of these two lines are identical, processing branches to line 4 of work file 3.

(4) Since the content of line 1 has now changed, processing does not branch to line 4 of work file 3.

Example 2

```
   1.     @SET #S4 = 'M' ------------------------------------------------ (1)
   1.     @PROC 4
   1.     @ @PRINT #S4 N ------------------------------------------------ (2)
   2.     @ @CREATE #S4: 'M',#S4
   3.     @ @IF #S4 < 'M'*8 : @GOTO 1
   4.     @END
   1.     @DO 4
M
MM
MMM
MMMM
MMMMM
MMMMMM
MMMMMMM
   1.
```

(1) The string variable #S4 contains the character M.

(2) The following procedure is entered in work file 4: the content of #S4 is to be output. This is to be followed by the current content of #S4 preceded by the letter M.

If the content of #S4 is smaller than MMMMMMMM then processing should start from the beginning 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 LUCK?
9.     @PROC 6
1.     @ @IF ! <> 'ABC' : @GOTO 3 ------------------------------------ (1)
2.     @ @CREATE !: '*' * 20
3.     @ @CONTINUE
4.     @END
9.     @DO 6,!=%,$ ---------------------------------------------------- (2)
9.     @PRINT
1.0000 ********************
2.0000 WHO
3.0000 ********************
4.0000 WANTS
5.0000 ********************
6.0000 TO TRY
7.0000 ********************
8.0000 HIS LUCK?
9.
```

(1) In work file 6, the line numbers are addressed via the loop counter `!`. If the content of the line currently addressed via the counter ! is different from `ABC` then the line content should not be modified. Otherwise, the line content is to be replaced by `********************` .

(2) Work file 6 is executed. During processing, all the lines in the current work file are to be addressed in sequence by the loop counter `!`.

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' ---------------------------------- (1)
   4.     @SET #I9 = 1
   4.     @PROC 1 ------------------------------------------------------- (2)
   1.     @ @IF %+#I9 = $-1L : @GOTO 4 ---------------------------------- (3)
   2.     @ @PRINT #S1 N
   3.     @ @RETURN
   4.     @ @PRINT #S0 N
   5.     @END
   4.     @DO 1 --------------------------------------------------------- (4)
RESULT POSITIVE
   4.     @SET #I9 = 2
   4.     @DO 1 --------------------------------------------------------- (5)
RESULT NEGATIVE
   4.
```

(1) The string variables #S0 and #S1 are filled with content. The value 1 is assigned to the integer variable #I9.

(2) Work file 1 is opened.

(3) If @DO 1 is subsequently issued then this line causes the line numbers %+#I9 and $-1L to be compared.

% addresses the first line number (i.e. 1).

$ addresses the last line number (i.e. 3).

$-1L addresses the penultimate line number (i.e. 2).

(4) The procedure in work file 1 is executed. At this point, the relation indicated there %+#I9=$-1L is true since 1+1=3-1 is true.

(5) At this point, the relation indicated in work file 1%+#I9=$-1L is false 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 ---------------------------------- (1)
2.     @ @CREATE $+1: 'OR PERHAPS NOT'
3.     @ @PRINT
4.     @END
4.     @$-1
2.     @DO 2 ------------------------------------------------------------ (2)
2.     @1
1.     @DO 2 ------------------------------------------------------------ (3)
1.0000 PLEASE DO NOT LAUGH
2.0000 AT THIS EXAMPLE
3.0000 IT IS TOO SIMPLE
4.0000 OR PERHAPS NOT
1.
```

(1) If when work file 2 is executed, the relation indicated here is not fulfilled (<> means not equal to) then the procedure is aborted at this point.

(2) Work file 2 is executed. Because `*=$-1=2`, the expression `%+6-#L3<>$-*` is equivalent to `1+6-5<>3-2` and is therefore true. The execution of the procedure is therefore aborted.

(3) At this point, `*=1` and consequently the relation in work file 2 `%+6-#L3<>$-*` is false because `1+6-5=3-1`. Consequently, the remaining statements in work file 2 are executed.

Example 6

```
   1.     @SET #I3 = 1 ------------------------------------------------- (1)
   1.     @PROC 7
   1.     @ @IF #I3 > 5 : @RETURN
   2.     @ @STATUS = #I3 ---------------------------------------------- (2)
   3.     @ @SET #I3 = #I3+1
   4.     @ @GOTO 1
   5.     @END
   1.     @DO 7 -------------------------------------------------------- (3)
#I03= 0000000001
#I03= 0000000002
#I03= 0000000003
#I03= 0000000004
#I03= 0000000005
   1.
```

(1) The value 1 is assigned to the integer variable #I3.

(2) The procedure in work file 7 should output the values for the integer variable #I3 (@STATUS =#I3) and increment these (#I3+1) until #I3 has a value greater than 5 for the first time.

(3) Work file 7 is executed.

9.59 @IF (format 3) – Query @ON hits or work file status

This format of the @IF statement makes it possible to check in EDT procedures whether EDT identified a hit the last time @ON was executed or whether the current work file is empty. Depending on the result, a specified string either is or is not processed as input.

Operation	Operands	L mode
@IF	{**.T**RUE**.** [rel col] / **.F**ALSE**.** / **.E**MPTY**.**} :[text]	

.TRUE. Processing branches if a hit was identified in the current work file the last time @ON was executed.

If `rel` and `col` are specified then the condition is structured in such a way that the column number in which the first identified hit begins is compared with the column number specified by `col` using the relational operator indicated by `rel`. The condition is only considered to be fulfilled if this comparison is positive.

rel Defines the relational operator for the column numbers (see above):

Symbol			Meaning
>	or	GT	greater than
<	or	LT	less than
>=	or	GE	greater than or equal to
<=	or	LE	less than or equal to
=	or	EQ	equal to
<>	or	NE	not equal to

col Column number which is compared with the number of the column in which the first hit started in the current work file when the last @ON statement was executed.

.FALSE. Processing branches if *no* hit was identified in the current work file the last time @ON was executed.

.EMPTY. Processing branches if the current work file is empty. A work file is empty if it contains no records

text EDT statement or data line. If the condition is fulfilled, the string is treated as if it had been entered at the prompt in L mode. In particular, the decision to interpret the text as data input or as a statement is made in accordance with the same rules (for more information, see section "L mode" on page 126).

The `text` operand starts immediately after the character `':'`, i.e. any specified blanks form part of the operand and are taken over into the line in the case of data input.

If `text` is not specified (although the colon is), then an empty line (line of length 0) is inserted.

The previous specification of `GOTO` or `RETURN` without a colon continues to be supported for reasons of compatibility.

Note

Using @IF with @RETURN as a statement outside of procedures may cause EDT to terminate (see the @RETURN statement).

Example 1

```
5.     @PRINT
1.0000 WHO
2.0000 WANTS
3.0000 TO TRY
4.0000 HIS LUCK
5.     @PROC 8
1.     @ @ON ! FIND 'I'
2.     @ @IF .FALSE. : @GOTO 4 ---------------------------------------- (1)
3.     @ @CREATE !: '*' * 20
4.     @ @CONTINUE
5.     @END
5.     @DO 8,!=%,$ ---------------------------------------------------- (2)
5.     @PRINT
1.0000 WHO
2.0000 WANTS
3.0000 TO TRY
4.0000 ********************
5.
```

(1) In work file 8, the line numbers are addressed via the loop counter `!`. If the letter `I` is not present in one of the addressed lines then the line remains unchanged. Otherwise, the line content is to be replaced by 20 asterisks.

(2) Work file 8 is executed. During processing, all the lines in the main file are to be addressed in sequence by the loop counter !.

Example 2

```
5.     @PRINT
1.0000 WHO
2.0000 WANTS
3.0000 PLENTY OF
4.0000 LUCK?
5.     @PROC 9
1.     @ @ON ! FIND 'EN'
2.     @ @IF .TRUE. = 3 : @GOTO 4
3.     @ @GOTO 5 ------------------------------------------------------ (1)
4.     @ @SUFFIX ! WITH ' GOOD'
5.     @ @CONTINUE
6.     @END
5.     @DO 9,!=%,$ ---------------------------------------------------- (2)
5.     @PRINT
1.0000 WHO
2.0000 WANTS
3.0000 PLENTY OF GOOD
4.0000 LUCK?
5.
```

(1) In the procedure in work file 9, the line numbers are addressed via the loop counter !. If the string EN occurs in columns 3 to 4 of one of the lines addressed in this way, then the string GOOD is to be appended to it. Otherwise, the line is left unchanged.

(2) The procedure in work file 9 is executed. During processing, all the lines in the main file are addressed in sequence by the loop counter !.

9.60 @IF (format 4) – Query job and user switches

In EDT procedures, this format of the @IF statement checks which job and/or user switches are active and inactive (see also @SETSW and section “Job switches” on page 98). Depending on the result, a specified string either is or is not processed as input.

Operation	Operands	L mode
@IF	{**ON** / **OF**F} = [**U**] int :[text]	

ON — Processing branches if the specified switch is set.

OFF — Processing branches if the specified switch is *not* set.

U — Specifies that a user switch is to be checked. If `U` is not specified then a job switch is checked.

int — Number of the switch (`0..31`) whose setting is to be checked.

If the keyword `U` is specified before the switch number then the user switch `int` belonging to the user's own ID is checked instead of the job switch `int`.

text — EDT statement or data line. If the condition is fulfilled, the string is treated as if it had been entered at the prompt in L mode. In particular, the decision to interpret the text as data input or as a statement is made in accordance with the same rules (for more information, see section “L mode” on page 126).

The `text` operand starts immediately after the character `':'`, i.e. any specified blanks form part of the operand and are taken over into the line in the case of data input.

If `text` is not specified (although the colon is), then an empty line (line of length 0) is inserted.

The previous specification of `GOTO` or `RETURN` without a colon in procedures continues to be supported for reasons of compatibility.

Note

Using @IF with @RETURN as a statement outside of procedures may cause EDT to terminate (see the @RETURN statement).

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             ---------------------------------------- (1)
   4.     @ @PRINT #S3 N
   5.     @END
   1.     @SETSW OFF = 15 ------------------------------------------------ (2)
   1.     @DO 8 ---------------------------------------------------------- (3)
SWITCH 15 IS OFF
   1.     @SETSW ON = 15 ------------------------------------------------- (4)
   1.     @DO 8 ---------------------------------------------------------- (5)
SWITCH 15 IS ON
   1.
```

(1) The procedure in work file 8 outputs the string variable #S3 if job switch 15 is set, and the string variable #S2 otherwise.

(2) Switch 15 is reset.

(3) The procedure in work file 8 is executed.

(4) Switch 15 is set.

(5) Work file 8 is executed.

9.61 @IF (format 5) – Query EDT parameter settings

This format of the @IF statement can be used in EDT procedures or in L mode to query the operating mode that is currently set (see section “Introduction to the EDT operating modes” on page 21). Depending on the result, a specified string either is or is not processed as input.

Operation	Operands	L mode
@IF	**OP**ERATING-MODE = { **U**NICODE / **C**OMPATIBL }	:[text]

OPERATING-MODE=
The EDT operating mode is checked.

UNICODE The condition is fulfilled if EDT is in Unicode mode.

COMPATIBLE
The condition is fulfilled if EDT is in compatibility mode.

text EDT statement or data line. If the condition is fulfilled, the string is treated as if it had been entered at the prompt in L mode. In particular, the decision to interpret the text as data input or as a statement is made in accordance with the same rules (for more information, see section “L mode” on page 126).

The `text` operand starts immediately after the character `':'`, i.e. any specified blanks form part of the operand and are taken over into the line in the case of data input.

If `text` is not specified (although the colon is), then an empty line (line of length 0) is inserted.

Note

Using @IF with @RETURN as a statement outside of procedures may cause EDT to terminate (see the @RETURN statement).

Example

```
1.     @IF OP = C : @GOTO 5
2.     @SET #S2 = 'PROCEDURE ONLY RUNS IN COMPATIBILITY MODE'
3.     @PRINT #S2 N
4.     @HALT
5.     @CONTINUE
6.     ...other statements
```

The procedure results in the abnormal termination of EDT if it is not running in compatibility mode.

9.62 @INDEX – Control line number display

In F mode, the @INDEX statement activates or deactivates the line number display for the current work file in the relevant data window (see also section “The work window” on page 103).

Operation	Operands	F mode
@INDEX	[{**ON** / **O**FF}]	

ON Activates line number display (default value).

OFF Deactivates line number display.

When an EDT session starts, the line number display is activated for both data windows of all work files.

If the line number display is activated in F mode then, depending on the employed terminal and the way it has been set with the @VDT statement (72 or 124 characters per line), the 6-digit line number display is output with a decimal point and a protected blank for the visual separation of the line contents.

If the line number display is deactivated then 80 or 132 characters are output per line.

In both formats, the first column of each line forms the statement code column.

The setting for the line number display is saved separately for the upper and lower (possible) data windows corresponding to each work file. If only one data window is displayed on the screen then the setting is made for both (possible) data windows. In contrast, if the screen is split (see @PAR SPLIT) then it applies only to the work window in which it was entered even if the same work file is displayed in both work windows.

Specifying @INDEX ON deactivates `EDIT-LONG` mode (see the @EDIT statement).

The @PAR INDEX statement can be used instead of @INDEX. Furthermore, @PAR INDEX can be used for a specific work file or globally for all the work files and is also permitted in L mode and therefore in EDT procedures. Please refer to the description of the @PAR statement for information on which of the data windows of the specified work file @PAR INDEX applies to.

Example

```
   1.00 EDT is the BS2000  file<·····································
   2.00 editor, used for the user-<·····································
   3.00 friendly creation and editing<·····································
   4.00 of BS2000 files in SAM and ISAM formats<·····························
@index off·····················································0001.00:00001(00)
```

The line number display is deactivated.

```
EDT is the BS2000  file<·····································
editor, used for the user-<·····································
friendly creation and editing<·····································
of BS2000 files in SAM and ISAM formats<·····························
```

The work window is displayed without line numbers.

9.63 @INPUT (format 1) – Start @INPUT procedure

This format of the @INPUT statement starts an @INPUT procedure from a file. The statements and/or records in the file are processed sequentially.

For information on the structure and processing of EDT procedures, see section "EDT procedures" on page 64.

Operation	Operands	F mode, L mode
@INPUT	**L**IBRARY=path1 ([**E**LEMENT=] elname [(vers)][,eltype]) **E**LEMENT=elname [(vers)][,eltype] **F**ILE = { path2 / *linkname } **P**OSIX-FILE=xpath [,**C**ODE=name]	[**P**RINT]

LIBRARY=... The @INPUT procedure is defined by explicitly specifying the library name and the element name.

path1 Name of the library.

elname Name of the element.

vers Version of the required element (see the LMS User Guide [14]). If `vers` is not specified or if `*STD` is specified then the highest available version of the element is selected.

eltype Type of element. Permitted type specifications are `S`,`M`, `P`, `J`, `D`, `X`, `*STD` as well as freely selectable type names having one of these types as basic type. If `eltype` is not specified then the default type specified with @PAR ELEMENT-TYPE is used. The permitted element types and their meanings are described in chapter "File processing" on page 131.

ELEMENT=... The @INPUT procedure is defined by the element name or by specifying the library name. The default library set with @PAR LIBRARY is used implicitly (if @PAR LIBRARY has been specified - otherwise the error message `EDT5181` is issued).

The operands `elname`, `vers` and `eltype` have the same meaning as when a library is specified explicitly (see above).

FILE= The @INPUT procedure is defined by the name of a BS2000 file.

path2 Name of the file that is to be read in as an @INPUT procedure.

*linkname
: File link name of the BS2000 file that is to be read in as an @INPUT procedure. The file name and the file attributes are stored in the `Task File Table`. The file link name must not be specified as the special file name `*BY-PROGRAM`. This results in the error `EDT4923`. If no file link name is defined then the statement is rejected with the message `EDT5480`.

  If the file link name is declared as the special file name `*DUMMY` then it is treated as an existing empty file.

POSIX-FILE=
: The @INPUT file is defined by the path name of a POSIX file.

  xpath
  : Path name of the POSIX file that is to be read in as an @INPUT procedure.

    The `xpath` operand can also be specified as a string variable. It must be specified as a string variable if the path name contains characters which have a special meaning in EDT syntax (e.g. blanks, semicolons in F mode or commas).

CODE=
: Defines the character set that is to be assumed for the POSIX file. Since it is not possible to assign character sets to POSIX files in the POSIX file system, a user specification is required here.

  If `CODE` is not specified then the character set defined in @PAR CODE is assumed.

  name
  : Character set of the POSIX file that is to be read in. The name of a valid character set must be specified for `name` (see section “Character sets” on page 47).

  EBCDIC
  : The keyword `EBCDIC` is now only supported for reasons of compatibility and is a synonym for the character set `EDF041`.

  ISO
  : The keyword `ISO` is now only supported for reasons of compatibility and is a synonym for the character set `ISO88591`.

PRINT
: Each line of the procedure should be logged before it is executed. In interactive mode, the output is written to `SYSOUT` and in batch mode it is written to `SYSLST`.
  Specifying `PRINT` also causes all error messages to be output and sets the EDT error switch. Normally, the two messages `EDT0901` and `EDT4932` are not output in procedures and the EDT error switch is not set (see section “Message texts” on page 638).

If the specified file does not exist or cannot be accessed as required or if the file cannot be read in successfully then the statement is rejected with a corresponding error message.

The @INPUT statement (format 1) must not be issued in @INPUT or in @DO procedures.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

The execution of an @INPUT procedure is terminated when the @RETURN statement or the last statement in the procedure has been executed.

If the @INPUT procedure contains records then these are inserted in the current work file in the same way data input in L mode (empty records are ignored). If the current work file is empty and has the character set *NONE then it is assigned the character set of the file if records are inserted. If this character set is *NONE then the work file is assigned the character set EDF03IRV when records are inserted.

The statements or records read from the @INPUT procedure are interpreted in the character set corresponding to the specified file. If this character set is *NONE then EDF03IRV is used.

This character set may differ from the character set used in the current work file. Since statements always apply to the current work file and records are always inserted in the current work file it may therefore be necessary to convert literals in statements or records. If the file contains characters which cannot be displayed in the work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the execution of the @INPUT procedure is aborted with the error message EDT5453.

If the file is present in a Unicode character set and contains an illegal byte sequence, e.g. surrogate characters, then it will be impossible to read it even if SUBSTITUTION-CHARACTERS is specified. In this case, the execution of the @INPUT procedure is aborted with the message EDT5454.

9.64 @INPUT (format 2) – Start @INPUT procedure from DMS file

This format of the @INPUT statement starts an @INPUT procedure from a SAM or ISAM file. Format 2 is now only supported for reasons of compatibility and should no longer be used. The statements and/or records in the file are processed sequentially. For information on the structure and processing of EDT procedures, see section "EDT procedures" on page 64.

Operation	Operands	F mode, L mode
@INPUT	file [(ver)] [lines[,...]] [:cols[,...]:] [{ **R**ECORD**S** / **K**EY }] [**P**RINT]	

file — Name of the SAM or ISAM file that is to be read and processed. The name must correspond to the SDF data type <filename 1..54>.

Here, the symbolic name '/' for a file for which the LINK name EDTSAM or EDTISAM has been assigned by means of the SET-FILE-LINK command is not permitted.

If the specified file does not exist or cannot be accessed as required or if the file cannot be read in successfully then the statement is rejected with a corresponding error message.

ver — Although this operand may be entered for the purposes of symmetry, it is completely ignored.

lines — One or more line ranges that are to be processed in the ISAM or SAM file. If lines is not specified then all the lines in the file are processed.

If symbolic line numbers are specified then their values are taken over from the current work file and therefore usually have nothing to do with the record structure of the specified file.
Any lines specification for SAM files is ignored unless one of the keywords KEY or RECORDS has been specified at the same time.

cols — One or more column ranges containing the statements to be processed. The ranges may repeat and overlap. If column values which exceed the record length are specified then blanks are read in their place.
If KEY is specified for SAM files or in the case of ISAM files then the column count starts after the key in the record.
If no column range is specified then the lines are read in full.

KEY
: Specifies that the first 8 characters of each line in a SAM file are to be interpreted as a line number. In the case of SAM files, this type of record can be created by specifying @WRITE together with the KEY operand.
If KEY is specified in the @INPUT statement then these numbers are interpreted as line content rather than line numbers when the file is read. Otherwise, EDT would consider every line in the file to be a text line.

RECORDS
: In the case of SAM files, specifies that a line range (see lines operand) is to be selected via the logical line number. The logical line number of the 1st line in the file is 0.0001, the logical line number of the 2nd line in the file is 0.0002 etc.

PRINT
: Each line of the procedure should be logged before it is executed. In interactive mode, the output is written to SYSOUT and in batch mode it is written to SYSLST.

  Specifying PRINT also causes all error messages to be output and sets the EDT error switch. Normally, the two messages EDT0901 and EDT4932 are not output in procedures and the EDT error switch is not set (see section "Message texts" on page 638).

The @INPUT statement (format 2) must not be issued in @INPUT or in @DO procedures.

The keywords KEY and RECORDS are ignored in the case of ISAM files. If neither RECORDS nor KEY is specified then any lines specification for a SAM file is ignored, i.e. all the lines in the file are processed.

The execution of an @INPUT procedure is aborted if the @RETURN statement or the last statement in the procedure has been processed.

In the case of ISAM files and SAM files (with the KEY operand), the record key should always contain a valid line number and, when processing SAM files, EDT expects the records to be present in ascending order.

If the file is read without any line range specification then the record keys are simply ignored, i.e. they are not checked. In contrast, if line ranges are specified then EDT processes SAM and ISAM files differently.
In the case of ISAM files, the keys corresponding to the records which are actually read are checked. If a record contains a non-numerical key then execution of the procedure is aborted with error EDT4984.
In the case of SAM files, records with non-numerical keys or with keywords which are smaller than the current lower range boundary are ignored. If the keyword is greater than the current upper range boundary then the record in question and all the records following it are ignored. Furthermore, in the case of SAM files, records that are shorter than 8 characters are also ignored.

If the @INPUT procedure contains records then these are inserted in the current work file in the same way data input in L mode (empty records are ignored). If the current work file is empty and has the character set *NONE then it is assigned the character set of the file if records are inserted. If this character set is *NONE then the work file is assigned the character set EDF03IRV when records are inserted.

The statements or records read from the @INPUT procedure are interpreted in the character set corresponding to the specified file. If this character set is *NONE then EDF03IRV is used.

This character set may differ from the character set used in the current work file. Since statements always apply to the current work file and records are always inserted in the current work file it may therefore be necessary to convert literals in statements or records. If the file contains characters which cannot be displayed in the work file's character set then these are replaced by a substitute character if such a character has been specified (see @PAR SUBSTITUTION-CHARACTER), otherwise execution of the @INPUT procedure is aborted with the error message EDT5453. This also applies if there are invalid characters outside of the column range that is to be read. In contrast, invalid characters outside of the line range that is to be read are ignored.

If the file is present in a Unicode character set and contains an illegal byte sequence, e.g. surrogate characters, then it will be impossible to read it even if SUBSTITUTION-CHARACTERS is specified. In this case, the execution of the @INPUT procedure is aborted with the message EDT5454.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
6.     @PRINT
1.0000 @DELETE
2.0000 I AM LINE 1
3.0000 I AM THE SECOND LINE
4.0000 @PRINT 1
5.0000 @PRINT 2
6.     @WRITE 'SAM-INP' KEY ------------------------------------------------ (1)
6.     @SAVE 'ISAM-INP' ---------------------------------------------------- (2)
6.     @INPUT 'SAM-INP' KEY ------------------------------------------------ (3)
1.0000 I AM LINE 1
2.0000 I AM THE SECOND LINE
3.     @INPUT 'ISAM-INP' 1-3,5,4 ------------------------------------------- (4)
2.0000 I AM THE SECOND LINE
1.0000 I AM LINE 1
3.
```

(1) The content of the work file is written as a SAM file. When this is done, each line is prefixed by a key calculated on the basis of the current line number.

(2) The content of the work file is written again, but in this case as an ISAM file.

(3) The complete file `SAM-INP` is read and executed. Since this file was created using @WRITE together with a `KEY` specification, `KEY` must be specified. Otherwise, the stored keys are not converted into line numbers.

(4) Lines `1-3,5,4` in the file `ISAM-INP` are to be read and executed in the specified sequence.

9.65 @INPUT (format 3) – Define EDT input mode

This format of the @INPUT statement allows users to define how EDT is to interpret text input in L mode.

Operation	Operands	L mode
@INPUT	[{ [**C**HAR] | **H**EX | **X** [**I**SO] | **B**INARY }]	

CHAR — Causes EDT to interpret record input in L mode as a sequence of characters in the character set applicable to the input source (terminal, SYSDTA, file, library element, work file) (see section "L mode" on page 126 which also discusses the handling of character sets during input in L mode).

HEX, X — Causes EDT to interpret record input in L mode as a sequence of hexadecimal characters (see section "L mode" on page 126 which also discusses the handling of character sets during hexadecimal input).

ISO — This operand is no longer supported in EDT V17.0 Unicode mode. For reasons of compatibility, it is ignored and no error message is issued if it is input. In EDT V17.0 Unicode mode, ISO character sets are not subject to any special processing. If an ISO character set is defined for the current work file then hexadecimal input for this work file is automatically interpreted in the correct code and there is no implicit conversion into EBCDIC.

BINARY — Causes EDT to interpret record input in L mode as a sequence of binary characters (see section "L mode" on page 126).

The default setting when EDT is called is @INPUT CHAR.

The maximum permitted abbreviation of the statement can only be used if operands are specified. If called without operands, the maximum permitted abbreviation is @INP as EDT otherwise recognizes the @INDEX statement.

Note

Even if @INPUT HEX or @INPUT BINARY has been specified, statements may *not* be entered in hexadecimal or binary coding.

9.66 @LIMITS – Output line numbers and number of lines

The @LIMITS statement causes EDT to output the lowest and the highest assigned line number as well as the number of lines for the current work file.

In interactive mode, the output is written in a line to SYSOUT and in batch mode it is written to SYSLST.

Operation	Operands	F mode, L mode
@LIMITS		

In the case of files opened for real processing, the number of lines 0 is always output.

Example

```
   4.     @PRINT
   1.0000 A
   2.0000 B
   3.0000 C
   4.     @LIMITS
   1.0000 TO    3.0000          3 LINES ---------------------------------- (1)

   4.     @COPY 1-3 TO 99.01 ------------------------------------------------ (2)

 100.03   @LIMITS
   1.0000 TO    99.0300         6 LINES ---------------------------------- (3)

 100.03
```

(1) The lowest and highest assigned line numbers are output together with the number of lines.

(2) Lines 1-3 are copied to the lines 99.01, 99.02 and 99.03.

(3) The lowest and highest assigned line numbers are now 1.0000 or 99.0300. The number of lines is now 6.

9.67 @LIST – Print work file ranges or string variables

The @LIST statement is used to output ranges of a work file or string variables to SYSLST or at the printer. Unless specified to the contrary, every output line is prefixed by the line number and every output string variable is prefixed by the number of the string variable.

Operation	Operands	F mode, L mode
@LIST	[{lines / svars} [:cols[:]] [**X**] [**N**] [{**C** [int] / **P** int}] [**I**] [**S**]] [,...]	

lines — The line range to be output.

svars — The range of string variables whose contents are to be output.

cols — Column range for output in the current work file or in the specified string variables.

If only one column number is specified then the remainder of the line or string variable is output as of this column. If the first column specification is greater than the length of the line or string variable then the line or string variable is ignored.

If no column range is specified then the line or string variable is read in full.

X — The lines are printed in hexadecimal form. The output format is the same as in @PRINT.. X.

N — The line numbers or the numbers of the string variables are omitted on printing.

C int — EDT expects an EBCDIC line feed character as the first character in each line in the specified column range. This generates a line feed during printing but is not itself printed. The line feed character must be present in the character set of the current work file or the current string variable (see below). EDT bases its interpretation of this character on the meaning of the equivalent EDF041 character.
Any characters which EDT cannot interpret in the first column result in a simple line feed.

Values between 0 and 256 are permitted for int. If a value of int other than 0 is specified, EDT generates not only the line feeds present in the record itself but also a form feed after int output lines (taking account of the form and line feeds present in the record itself). If int has the value 0 then no additional form feed is generated.

The value set here for the page size remains valid and therefore influences all outputs to SYSLST.

If int is not specified, EDT uses the last page size setting defined with the @LIST or @PAGE statement independently of whether int was specified in combination with C or P. If there has been no previous @LIST modifying the value of int then the default value 65 is assumed (see @PAGE statement).

P int

EDT prefixes every output record with an EBCDIC line feed character. When doing this, EDT only uses the EBCDIC feed characters Line feed after printing each line or Form feed in the coding corresponding to the character set for the output file in question (SYSLST or temporary file) (see below).

Values between 0 and 256 are permitted for int. If int is not equal to 0 then a form feed is inserted exactly after every int lines. If int has the value 0 then no form feed is inserted. The value set here for the page size remains valid and therefore influences all outputs to SYSLST.

I

Printing starts immediately. This means that EDT writes the records that are to be output to a temporary file (using the system file SYSLST97) and then outputs these after closure to the defined standard printer by means of a /PRINT-DOCUMENT command.

When ranges from a work file are output, the temporary file is created in the character set used by the work file provided that this is an EBCDIC character set. If not, the reference character set is used provided that this is an EBCDIC character set. In all other cases, the temporary file is created in the character set UTFE. If ranges of string variables are to be output then the character set is determined in the same way for each individual string variable. If the character sets determined in this way are all identical then this character set is used, otherwise UTFE.

If the user is not authorized to create temporary files or if the system administrator has prohibited the use of temporary files or if, for any other reason, it is not possible to create the file or write its content then a @LIST statement issued with the I operand is rejected with a corresponding error message.

The I operand is only permitted in interactive mode. It is only of value if the defined standard printer is able to reproduce the character set used for the output correctly.

If I is not specified then the output is sent to SYSLST. If SYSLST is not assigned to a file then printing does not start until after /LOGOFF. If SYSLST is assigned to a file then users must initiate output themselves.

S Eliminates the additional line feed which usually takes place before the first output line.

If neither `lines` nor `svars` is specified then the entire current work file is output.

If neither `P` nor `C` is specified, EDT generates feed characters as in the case of `P` and the last page size value set using @LIST or @PAGE is used.

If EDT generates the feed characters then it takes account of the set or specified page size and usually generates an additional line feed before the first line of every range. This operation is only omitted if the `S` operand is specified or if this line is output at the start of a page. A line feed is inserted in the output after every 132 characters (or 160 characters if job switch 6 is set).

When ranges in a work file are output, these are converted from the character set used in the work file into the character set used for `SYSLST` or the character set of the temporary file (see the description of the `I` operand. When ranges of string variables are output, each string variable is converted from the character set assigned to it into the character set used for `SYSLST` or the character set of the temporary file (see the description of the `I` operand. If characters are found which do not correspond to a valid character in the target character set then these are replaced by a substitute character if such a character has been specified (see @PAR SUBSTITUTION-CHARACTER). Otherwise blanks are used.

The EBCDIC and ASA feed characters in BS2000 are valid characters in every employed 8-bit or Unicode character set and can therefore always be edited in EDT even if they cannot be displayed. This is particularly true of the feed characters which are interpreted or inserted by EDT (see the `C` and `P` operands. As a result, some feed characters are coded as Unicode characters with more than one byte (see the table below).

In the case of print files which are present in a Unicode character set (e.g. SYSLST), the BS2000 SPOOL subsystem converts the feed character from the print file's Unicode character set into `EDF041` and interprets the resulting character as explained in the User Guide, Commands, Volume 3 [10], description of the `/PRINT-DOCUMENT` command. Conversion is only performed if the file is printed using `LINE-SPACING=*BY-EBCDIC-CONTROL` or `LINE-SPACING=*BY-ASA-CONTROL`. In the case of files that are to be printed using `LINE-SPACING=*BY-IBM-CONTROL` no conversion is performed. In these files, the feed characters are not usually valid Unicode characters with the result that they cannot be processed in EDT. If the print file possesses a 7-bit or 8-bit character set then, as in the past, SPOOL interprets the feed character without converting it.

The codings of the EBCDIC feed characters interpreted or generated by EDT are given below for a number of character sets:

UTF16	UTF8	UTFE	EDF041	Character	
0020	20	40	40	␣	No feed before printing, new line after printing
00a0	c2a0	6741	41		One line feed before printing, new line after printing
00e2	c3a2	68b0	42	â	Two lines feed before printing, new line after printing
00e4	c3a4	689f	43	ä	Three lines feed before printing, new line after printing
00e0	c3a0	6841	44	à	Four lines feed before printing, new line after printing
00e1	c3a1	68aa	45	á	Five lines feed before printing, new line after printing
00e3	c3a3	68b1	46	ã	Six lines feed before printing, new line after printing
00e5	c3a5	68b2	47	å	Seven lines feed before printing, new line after printing
00e7	c3a7	68b5	48	ç	Eight lines feed before printing, new line after printing
00f1	c3b1	688f	49	ñ	Nine lines feed before printing, new line after printing
0060	60	4a	4a	`	Ten lines feed before printing, new line after printing
002e	2e	4b	4b	.	Eleven lines feed before printing, new line after printing
003c	3c	4c	4c	<	Twelve lines feed before printing, new line after printing
0028	28	4d	4d	(	Thirteen lines feed before printing, new line after printing
002b	2b	4e	4e	+	Fourteen lines feed before printing, new line after printing
007c	7c	4f	4f	|	Fifteen lines feed before printing, new line after printing
0041	41	c1	c1	A	Page feed before printing

When interpreting feed characters, EDT also accepts the EBCDIC control characters x'00'..x'0F' (or their equivalents in other character sets) and interprets them as x'40'..x'4F'. However, when output is written in the character set UTF8 or UTFE, EDT never generates feed characters that are coded with more than one byte. Instead it generates the corresponding number of single line feeds.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Note

The system files SYSLST and SYSOUT should only be assigned to files or library elements with a Unicode character set if it is certain that only EDT sends output to these files. Otherwise files containing invalid characters could be created since other system components do not usually take account of the character set assigned to SYSLST or SYSOUT.

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 ------------------------------------------------------------ (1)
6.     @LIST 4-5 N ------------------------------------------------------ (2)
6.     @LIST 4 :13-14 X ------------------------------------------------- (3)
6.     @LIST & I -------------------------------------------------------- (4)
```

(1) The entire content of the work file is to be output to SYSLST. The print-out (possibly initiated by the system) is not started until LOGOFF.

Print 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.
```

(2) Lines 4 to 5 are to be output to SYSLST without line numbers.

Print output

```
TRANSFERRED TO PAPER
IN ANY DESIRED FORM.
```

(3) The two columns 12 and 13 of line 4 are to be output in hexadecimal to SYSLST.

Print output

```
4.0000 E3D6
```

(4) All the lines are to be printed immediately.

Print output

As in (1). However, the following system message is also displayed:

```
%  SCP0810 SPOOLOUT OF FILE 'XXX' ACCEPTED, TSN: 'XXX', PNAME: 'XXX'.
```

in order to confirm that the print job has been assigned.

9.68 @LOAD – Load program

The @LOAD statement terminates the EDT session and loads the specified program.

Operation	Operands	F mode, L mode
@LOAD	string	

string — String specifying the name of the program that is to be loaded. The system expects the name of a BS2000 file which contains the program that is to be loaded. It is not possible to specify a library element.

The @LOAD statement is one of the EDT statements with security implications (see also section "Access protection" on page 99). The statement is rejected in uninterruptible system procedures in interactive mode and on input from a file (read with `RDATA` from `SYSDTA` not equal to `SYSCMD`, execution of a start procedure).

The @LOAD statement always causes EDT to be terminated irrespective of whether the specified program file exists or contains a valid program.

As far as the handling of unsaved files and the related security queries is concerned, @LOAD acts in the same way as the @HALT statement (see section "Terminating an EDT session" on page 92). Since EDT is always terminated, save queries may, unlike in the case of @HALT, also be issued if the statement was entered in the screen dialog (started with @DIALOG).

If EDT was loaded as a subroutine and the EDT screen dialog has been activated with @DIALOG, the @LOAD statement does not result in the continuation of the user program. Instead, the user program is also unloaded.

It is therefore possible to prohibit users from issuing the @LOAD statement when EDT is called as a subroutine. In this case, calls are rejected with the message `EDT4976`.

Note

If @DIALOG was issued in a system procedure, then the remaining procedure commands after @LOAD are executed while the specified program is loaded instead of EDT. This may result in unwanted effects.

Example

The example assumes that the records present in the work file have not yet been saved.

```
   1.00 EDT is to be terminated<·····································
   2.00 and LMS loaded<··············································
   3.00 This is done with the @LOAD statement<·····················
   4.00 ·····························································
:                                                                :
:                                                                :
:                                                                :
@LOAD '$lms'·····················································0001.00:00001(00)
```

EDT is to be terminated and LMS is to be loaded.

```
%  EDT0900 EDITED FILE(S) NOT SAVED!
 LOCAL FILE ( 0) :
%  EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)?y
%  BLS0500 PROGRAM 'LMS', VERSION 'V3.0A1 OF 'yy-mm-dd' LOADED,
/resume-program
%  LMS0310 LMS VERSION V03.0A00 LOADED
CTL=(CMD)                                                    PRT=(OUT)
$
```

Since the work file has not yet been saved, EDT queries (in the same way as in @HALT), whether it should really terminate.

Since @LOAD was specified rather than @EXEC, a slash indicates that further system commands are expected. LMS is not started until the `/RESUME-PROGRAM` command is issued.

9.69 @LOG – Control logging

The @LOG statement controls the logging of the input in batch and interactive mode.

Operation	Operands	F mode, L mode
@LOG	[{**A**LL | **C**OMMANDS | **N**ONE}] [{**SYSLST** | **SYSLST** n}]	

ALL — All L mode input (text and statements) that is entered via `RDATA` or via the terminal is to be logged.

In the case of inputs in the F mode's interactive mode, the input in the statement lines (separated into individual statements if the case of statement sequences) is logged.

COMMANDS — Only statements are to be logged.

NONE — Nothing is to be logged.

SYSLST — Log output is sent to `SYSLST`. This is the default setting when EDT is started.

SYSLST n — Log output is written to the file to which `SYSLSTnn` is assigned (values between `1` and `99` are permitted for `n`).

The default setting when EDT is started in batch mode is @LOG NONE if job switch 4 is set and @LOG COMMANDS if job switch 4 is not set. If EDT is called in interactive mode then @LOG NONE is set by default.

The definition of the output medium (`SYSLST`, `SYSLSTnn`) remains valid for subsequent @LOG statements unless these specify a different value.

Note

Statements and data input which are read from EDT procedures and executed are not logged by @LOG in either batch or interactive mode. The logging of such items must be requested explicitly by means of @DO ...PRINT or @INPUT ...PRINT.
In test mode, the @LOG statement is not just checked for its syntax. It is also executed (see @SYNTAX- statement).

Logging to list variables, which was possible in the predecessor version, is no longer supported. It is, however, possible to use the `/ASSIGN-SYSLST` command to assign a list variable to the system file `SYSLST`.

9.70 @LOWER – Lowercase and uppercase on input

The @LOWER statement specifies whether or not EDT is to convert lowercase characters into uppercase when data and statements are input at the terminal.

Operation	Operands	F mode, L mode
@LOWER	[{**ON** / **OF**F}]	

ON — EDT differentiates between uppercase and lowercase. Strings are processed in the form that they are entered.

OFF — EDT converts entered lowercase characters into uppercase.

In F mode, any lowercase characters present in the work file are converted into smudge characters for output in the work window. If output is sent to SYSOUT or SYSLST (e.g. by means of the @ON statement, format 1, in L mode) then they are displayed as printable characters.

When EDT starts, the value ON is set as the default for all the work files.

The @LOWER statement applies globally to all the work files. The @PAR LOWER statement can be used to set the way in which lowercase characters are handled for each work file separately.

EDT uses the system component XHCS when converting from lowercase to uppercase. Which characters are converted therefore depends on the definition of the associated character set attributes in XHCS.

If the @LOWER statement is issued in L mode inside an input block (see @BLOCK) or in F mode as part of a statement sequence (statements separated by ;) then the conversion mode takes effect as of the statement or data line that follows @LOWER.

If @LOWER OFF is set then all the characters entered at the terminal are converted from lowercase to uppercase irrespective of whether the input was made in F mode or L mode or whether the input consists of statements or data lines. However, in F mode, this conversion is not performed until a statement is stored in the statement buffer, i.e. statements are stored here in the same way that they were entered (see also @SHIH statement).

If input is read from files or work files, e.g. during the execution of EDT procedures or when reading from SYSDTA after this has been assigned to a file, then text input and literals in statements are not converted into uppercase even if @LOWER OFF is set. This is the same behavior as when reading from files in which case there is also no conversion.

When statements are input, the setting @LOWER ON only affects literals. Statements and keywords are always converted into uppercase when a statement is analyzed.

9.71 @MODE – Change operating mode

The @MODE statement is used to switch between the operating modes (compatibility mode and Unicode mode, see section “Introduction to the EDT operating modes” on page 21).

Operation	Operands	F mode, L mode
@MODE	**OP**ERATING-MODE = { **U**NICODE / **C**OMPATIBL }	

OPERATING-MODE=
The EDT operating mode is switched.

UNICODE EDT changes from compatibility mode to Unicode mode. If EDT is already running in Unicode mode, the statement is ignored.

COMPATIBLE
EDT changes from Unicode mode to compatibility mode. If EDT is already running in compatibility mode, the statement is ignored.

It is only possible to change operating mode if all the EDT work files are empty and no files are open. Otherwise, the statement is rejected with the message EDT4983.

Changing the operating mode amounts to terminating EDT in one mode and then restarting it in another mode. When this is done, all the settings are lost and all the variables are reinitialized. For details, see section “Activating compatibility and Unicode mode” on page 615.

9.72 @MOVE – Move lines or string variables

The @MOVE statement transfers records from the current or another work file to the current work file and then deletes them at their original positions or transfers the contents of string variables to the current work file and then reinitializes the string variables.

For the sake of clarity, the line range in the source work file which contains the records that are to be moved or the range of string variables are referred to as the “source range” below. The line range in the current work file into which the records from the source work file are to be moved is referred to as the “target range”.

Operation	Operands	F mode, L mode
@MOVE	{ { lines [(procnr)] / svars } [**T**O {line1 [(inc)] [:] [line2]} [,...]] } [,...]	

lines — Contiguous line range that is to be moved to the current work file. Symbolic line numbers in lines refer to the line numbers of the current work file even if the lines are taken over from another work file.

procnr — Number of the source work file from which the records are to be moved (`0..22`). If `procnr` is not specified then the records are moved from the current work file. An active work file may not be specified. If the `TO` operand is not specified then `procnr` must not be the current work file.

svars — Range of string variables whose contents are to be moved into the current work file. After transfer, the string variables are deleted, i.e. they are reinitialized with a blank in the character set `EDF041`.

TO... — The operands which follow `TO` define the target range or ranges. If no target range is specified then the line numbers in the source work file are taken over into the current work file.

If the source work file is the current work file or if string variables are moved then `TO...` must be specified. In these cases, if no target range is specified then the @MOVE statement is rejected with the error message `EDT3218`.

line1 — Number of the first line in the target range.

inc — Increment used to form the line numbers following `line1`. If `inc` is not specified then the increment implicitly specified by `line1` is used (see section “Implicit increment assignment” on page 35).

: — The operands `line1` and `line2` should be separated by `:` if `inc` is not specified.

line2 Specifies the largest possible line number in the target range up to which the transfer of records is permitted.

As a result, no move operation is performed to lines in the current work file with line numbers higher than `line2`. This also applies if it is not possible to move all the records in the source range to the target range.

If `line2` is not specified then the @MOVE statement does not define any maximum value for the line numbers in the target range.

In the @MOVE statement, it is possible to specify multiple comma-separated source ranges each of which are associated with multiple target ranges. The number of source and target ranges is only limited by the maximum permitted length of EDT statements. It is not usually of value to specify multiple target ranges since the lines are deleted in the source range the first time they are transferred and are therefore no longer available for further move operations.

If the source and target ranges overlap then the source range is moved and deleted line-by-line.

Any existing lines with the same line numbers present in the work file are overwritten on the move operation.
If a line with a number greater than the previous highest line number is created then the current line number is modified.

If the current work file is empty and has the character set `*NONE` then it is assigned the character set of the source work file or the first specified string variable when the move operation is performed.

If the current work file has a character set then the lines to be moved or the contents of the string variables are converted into the character set of the current work file. If characters which cannot be displayed in the work file's character set are identified then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @ MOVE statement is rejected and error message `EDT5453` is output.

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

Note

Since the above syntax permits the omission of the `TO` operand, it is not always possible to distinguish unambiguously between the target and source ranges. In such cases, EDT interprets the ambiguous specification as a target range. Thus, for example, in the input

```
@MOVE 2-3(1) TO 7,1(1)
```

the specification `1(1)` is interpreted as a second target range (the `1` in parentheses is interpreted as the increment), whereas the specification `1(0)` at this point would be interpreted as the next source range (the `0` cannot be an increment and is interpreted as a work file number). If, in this example, the user wants to force the specification to be interpreted as a source range, it would be possible, for example, to enter

```
@MOVE 2-3(1) TO 7,1-1(1)
```

to eliminate all ambiguities.

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<..............................................................
  90.00 THE LINE IS NEVER OVERWRITTEN<.............................................
  91.00 .....................................................................................
 
 
 
 
move 2-4 to 20.....................................................0001.00:00001(00)
```

Lines 2 to 4 are to be moved to the line range starting at line 20. The value 1 is specified implicitly as the increment for the target range.

```
   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 THE LINE IS NEVER OVERWRITTEN<.............................................
  91.00 .....................................................................................
 
 
 
 
move 20-22 to 100 (5).............................................0001.00:00001(00)
```

Lines 20, 21 and 22 have now been created with the implicit increment 1 and lines 2, 3 and 4 have been deleted.

Lines 20-22 are to be copied to 100, 105 and 110.

```
   1.00 THIS LINE IS NOT MOVED<·····································
  90.00 THE 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·····························0001.00:00001(00)
```

The line range from line 100 through to the end of the work file (`100-.$`) is to be copied to the line range starting at line 82 with the explicit increment 5. Since 89 has been specified as the highest possible line number for the target range, line 90 is not written.

```
   1.00 THIS LINE IS NOT MOVED<·····································
  82.00 LINE 2 AND LINE 3<·····································
  87.00 AND LINE 4 ARE MOVED<··································
  90.00 THE LINE IS NEVER OVERWRITTEN<·····························
 110.00 SEVERAL TIMES<·····································
 111.00 ·····································
```

Since 89 has been specified as the highest possible line number for the target range, line 110 is not transferred.

9.73 @NOTE – Empty statement

The @ NOTE statement does not perform any action. It is used to insert comments in EDT procedures. Lines which contain a @NOTE statement can also be branched to by means of @GOTO. The @ CONTINUE statement offers the same functionality as @ NOTE.

Operation	Operands	L mode
@NOTE	[comment]	

comment The `comment` operand may contain any text as a comment.

Alongside the insertion of comments, this statement is also frequently used to define a last line in an EDT procedure which can be specified as the destination of a branch operation in a @GOTO or an @IF statement. This construction is required if an EDT procedure is called in an external loop with a loop counter (e.g. @DO 5,!=%,$), and an @IF ... RETURN would result in an unwanted abort of the external loop. Instead, processing branches 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
5.0000 PROCEDURE AT A TIME
6.     @PROC 1
1.     @1.00
1.00   @ @NOTE OBJECTIVE: IF A LINE CONTAINS A 'W' ------------------- (1)
1.01   @ @NOTE            DISPLAY IT ON THE SCREEN
1.02   @ @ON ! FIND 'W'
1.03   @ @IF .FALSE. : @GOTO 2
1.04   @ @PRINT !
1.05   @2.00
2.00   @ @NOTE ------------------------------------------------------ (2)
2.01   @END
6.     @DO 1,!=1,$ ---------------------------------------------------- (3)
1.0000 WITH EDT
2.0000 ANYONE WHO KNOWS
4.0000 WRITE HIS PROGRAM
6.
```

(1) In this case, @NOTE is used to insert a comment.

(2) In this case, @ NOTE is required because there must be a last line in a procedure that can be branched to.

(3) @DO with a loop counter executes the procedure in work file 1 which acts on work file 0.

9.74 @ON (format 1) – Output lines or string variables containing the search term

This format of the @ON statement causes EDT to output the content of each line or string variable in which a hit is found. In interactive mode, the output is written to SYSOUT and in batch mode it is written to SYSLST.

Operation	Operands	F mode, L mode
@ON	{ lines / svars } [,...] [:cols[:]] **P**RINT [**A**LL] [**F**] [**R**] [**NOT**] [**P**ATTERN] search [,int] [**S**] [**N**] [**E**]	

lines — One or more line ranges in which the search is to be performed.

svars — One or more ranges of string variables in which the search is to be performed.

cols — Contiguous column range to which the search is to be limited.

If the range specification contains only a single column specification, this indicates the range from the specified column through to the end of the line. If the first column specification is greater than the line length then the line or string variable is ignored.

If no column range is specified then the column range specified with @SEARCH-OPTION is used.

ALL — The ALL operand only has an effect if the E operand is also specified. In this case, all the hits in a line are highlighted. If ALL is not specified but E is, then only the first hit in a line is highlighted.

F — Only the first hit line in each specified line range is output. If F is not specified then all the hit lines in each specified line range are output. If both F and E are specified then the first hit in the first hit line in each specified line range is highlighted. If F, ALL and E are specified then all the hits in the first hit line in each specified line range is highlighted.

R — The lines are searched through from right to left. If R is not specified then they are searched through from left to right.

If R is specified but neither ALL nor F is then the minimum possible abbreviation for PRINT is PR.

NOT
: A hit is identified if the search term is not present in the specified column range in a line (negative search). In this case, the `ALL` and `E` operands are meaningless.

PATTERN
: The wildcards present in the search term are interpreted.

search
: Search term that is to be searched for in the search range (for details, see section "Searching with @ON" on page 78). It is not permissible to specify an empty string.

int
: Only the `int`th occurrence of the search term in a line is considered to represent the first hit. Values between 1 and 32768 are permitted for `int`. The default value for `int` is 1 byte.

S
: The empty line which precedes the first line for output is omitted. This operand only has an effect if the output is sent to `SYSLST`. If `S` is not specified then the output begins with an empty line.

N
: The hit lines are output without an associated line number. Equally, no string variable names (`#S00..#S20`) are output if hits are searched for in string variables. If `N` is not specified then the line numbers and the names of the string variables are output.

E
: The hits are highlighted when the hit lines are output on screen. If both `E` and `ALL` are specified then all the hits in the hit line are highlighted, otherwise only the first hit. If `E` is not specified then the hits are not highlighted on output.

  This operand is only effective if `VTCSET ON` is set; otherwise it is ignored. When long lines are output, the system may interrupt the operation with a `%PLEASE ACKNOWLEDGE`. If this occurs within a highlighted hit string then the remainder of the string, which is displayed in the following screen, is not highlighted.

  If inserting the terminal control characters in the line that is to be output would increase its length beyond 32768 characters then the insertion operation is aborted and the hit strings in the remainder of the line are not highlighted. The message `EDT1248` informs the user of this.

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 THEN NOTHING<·····················································
   6.00 IS DISPLAYED.<··························································
   7.00 ·····································································
:
:
:
on & print f 'HIT'
```

The first line containing the string `HIT` is to be output.

```
   2.0000 IN 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 THEN NOTHING<·····················································
   6.00 IS DISPLAYED.<··························································
   7.00 ·····································································
:
:
:
on & print 'HIT';create 100: #l0:#i0-#i1:·····················
```

All the lines containing the string `HIT` are to be output. The first located hit string is then to be written to line 100.

```
   2.0000 IN WHICH A HIT OCCURS
   5.0000 HIT THEN NOTHING
%PLEASE ACKNOWLEDGE
```

If there is more than one hit then the contents of the line number variable `#L0` and the integer variables `#I0` and `#I1` apply to the first detected hit, i.e.. `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 THEN NOTHING<·····················································
   6.00 IS DISPLAYED.<·························································
 100.00 HIT<·····························································
 101.00 ·····························································
```

```
on &:2 print 'HIT'·····························0001.00:00001(01)
```

All the lines which contain the string `HIT` after column 2 are to be output.

```
    2.0000 IN 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 THEN NOTHING<·····················································
   6.00 IS DISPLAYED.<·························································
 100.00 HIT<·····························································
 101.00 ·····························································
```

```
on & print 'UT',2·····························0001.00:00001(01)
```

The lines which contain the string `UT` at least twice are to be output.

```
    3.0000 ARE TO BE OUTPUT.
 %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 THEN NOTHING<·······················································
   6.00 IS DISPLAYED.<·······················································
 100.00 HIT<·································································
 101.00 ······································································
```

```
on & print not HIT·····································0001.00:00001(01)
```

All the lines which do not contain the string `HIT` are to be output.

```
    1.0000 ALL LINES IN
    3.0000 ARE TO BE OUTPUT.
    4.0000 IF THERE IS NO
    6.0000 IS DISPLAYED.
 %PLEASE ACKNOWLEDGE
```

9.75 @ON (format 2) – Output the start column of a hit string

This format of the @ON statement causes EDT to output the line numbers and the numbers of the columns in which hit strings begin when searching in work files. In interactive mode, the output is written to SYSOUT and in batch mode it is written to SYSLST. If no search term is specified in the statement, EDT outputs the line numbers and the length of each line in the specified line range.

When the search is performed in string variables, the names and lengths of the string variables are output instead of the line numbers and line lengths. The length specifications indicate the number of characters in the lines or string variables.

Operation	Operands	F mode, L mode
@ON	{ lines / svars } [,...] [:cols[:]] **CO**LUMN [[**A**LL] [**F**] [**R**] [**P**ATTERN] search [,int]]	

lines — One or more line ranges in which the search is to be performed.

svars — One or more ranges of string variables in which the search is to be performed.

cols — Contiguous column range to which the search is to be limited.

If the range specification contains only a single column specification, this indicates the range from the specified column through to the end of the line. If the first column specification is greater than the line length then the line or string variable is ignored.

If no column range is specified then the column range specified with @SEARCH-OPTION is used.

ALL — Each time a hit string is found in a line, the value of the column at which the string begins is output. If ALL is not specified then only the column value corresponding to the first hit string in a line is output.

F — Only the column values for the first hit line in each specified line range are output. If F is not specified then the column value for each hit line in each line range is output.

R — The lines are searched through from right to left. If R is not specified then they are searched through from left to right.

PATTERN — The wildcards present in the search term are interpreted.

search — Search term that is to be searched for in the search range (for details, see section "Searching with @ON" on page 78). It is not permissible to specify an empty string.

int — Only the `int`th occurrence of the search term in a line is considered to represent the first hit. Values between 1 and 32768 are permitted for `int`. The default value for `int` is 1 byte.

The output column values have five digits since the maximum number of characters that can be accommodated in a work file line or a string variable is 32768. A line of output may contain up to 10 column specifications followed by a line feed. The line number is not output in the continuation line.

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·····················································0001.00:00001(01)
```

The lengths of all the lines are to be output.

```
    1.0000 00020
    2.0000 00012
    3.0000 00032
 %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 1 column r 'E '·····································0001.00:00001(01)
```

The column at which the string 'E ' occurs for the first time (searching from the right) is to be output.

```
   1.0000 00016
%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 1 column all 'E '···································01.00:00001(01)
```

All the columns in which the string 'E ' occurs in line 1 are to be output.

```
   1.0000 00016
%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 'E ',3·····································0001.00:00001(01)
```

The column number of the hit (i.e. third occurrence of the search term E) is to be output for all the lines which contain at least three occurrences of the string E.

```
   3.0000 00028
%PLEASE ACKNOWLEDGE
```

```
   1.00 HOW LONG IS LINE 1 ?<·····················································
   2.00 AND LINE 2 ?<·····························································
   3.00 WHO KNOWS THE LENGTH OF LINE 3 ?<···v·····································
   4.00 ··························································································
```

```
on & column all 'E'·····································001.00:00001(01)
```

All the lines and column numbers which contain the search term are to be output.

```
   1.0000 000016
   2.0000 00008
   3.0000 00013 00016 00028
%PLEASE ACKNOWLEDGE
```

9.76 @ON (format 3) – Mark lines with search term

This format of the @ON statement causes all records in which a hit is identified to be flagged with the specified record mark. In F mode, the work window is positioned at the first hit record.

Operation	Operands	F mode, L mode
@ON	{ lines / svars } [,...] [:cols[:]] **F**IND [**A**LL] [**F**] [**R**] [**NOT**] [**P**ATTERN] search [,int] [**M**ARK [m]]	

lines — One or more line ranges in which the search is to be performed.

svars — One or more ranges of string variables in which the search is to be performed.

cols — Contiguous column range to which the search is to be limited.

If the range specification contains only a single column specification, this indicates the range from the specified column through to the end of the line. If the first column specification is greater than the line length then the line is ignored.

If no column range is specified then the column range specified with @SEARCH-OPTION is used.

ALL — Although it is permissible to specify `ALL` the specification is pointless since a record can only be marked once.

F — Only the first hit line in each line range is marked. If `F` is not specified then each hit line in each specified line range is marked.

R — This specification is pointless since whether or not hits are found in a line is independent of the direction of the search.

NOT — A hit is identified if the search term is not present in the specified column range in a line (negative search).

PATTERN — The wildcards present in the search term are interpreted.

search — Search term that is to be searched for in the search range (for details, see section “Searching with @ON” on page 78). It is not permissible to specify an empty string.

int — Only the `int`th occurrence of the search term in a line is marked. All values between 1 and 32768 are permitted for `int`. The default value for `int` is 1 byte.

MARK — The hit lines are marked. If the operand is not specified, the hit lines are assigned the mark 1. If string variables are specified in the search range then the `MARK` operand is ignored for these.

m — Number of the mark (`1..9`). If `m` is not specified, the hit line is assigned record mark 1.

Any existing record marks (e.g. those set by previous @ON statements) are retained.

If the statement is used for a file opened for real processing with @OPEN (format 2) then no records are marked. In F mode, the work window is simply positioned at the first hit record. The explicit specification of `MARK` or `MARK m` is rejected with the error message `EDT4935`.

If the statement is interrupted with [K2] and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

Note

The statement @ON lines FIND NOT PATTERN '/'... can be used to specifically mark empty lines (records of length 0) if `'/'` is the wildcard that stands for precisely one character.

Example

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<......................
   2.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<......................
   3.00 DUCK     DONALD    WALTSTR.8      DISNEYLAND<..........................
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<......................
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<......................
   6.00 ...................................................................... 




on & find 'STR.' mark 2.......................................0001.00:00001(01)
```

The records which contain the search term `STR.` should be flagged with record mark 2. EDT automatically moves to the 1st hit record.

```
   3.00 DUCK     DONALD    WALTSTR.8      DISNEYLAND<·····························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·························
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·························
   6.00 ···································································
:                                                                                 :
:                                                                                 :
:                                                                                 :
+(2)·····················································.0003.00:00001(01)
```

The work window has been positioned at line 3 since this contains the first hit record.

`+(2)` is specified to scroll to the next record with record mark 2.

```
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·························
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·························
   6.00 ···································································
```

9.77 @ON (format 4) – Copy marked lines

This format of the @ON statement copies all the records marked with the specified record mark in the searched line ranges into the specified work file.

Operation	Operands	F mode, L mode
@ON	lines[,...] [:cols[:]] **F**IND [**A**LL] [**F**] [**R**] [**NOT**] **M**ARK m [**C**OPY [**T**O]] (procnr) [**K**EEP] [**O**LD]	

lines — One or more line ranges in which the search is to be performed.

cols — Although the column range may be specified, this has no significance since only record marks are searched for.

ALL — Although the specification is permitted, it is pointless since a record can only be marked once.

F — Only the first hit record with the specified record mark in each specified line range is copied. If `F` is not specified then all the records with the specified record mark in each specified line range are copied.

R — This specification is of no significance since the search direction is irrelevant when searching for record marks.

NOT — If `NOT` is specified then records which possess a record mark other than the specified record mark `m` are copied. If `NOT` is not specified then records which possess the record mark `m` are copied.

This format cannot be used to copy records which do not have a record mark.

m — Number of the record mark (`1..9`) that is to be searched for.

procnr — The number of the work file (`0..22`) to which the hit lines are to be copied.

An active work file or the current work file may not be specified. If `OLD` is not specified and hit records are found then the target work file is deleted in full before the copy operation (see @DELETE, format 2). If no hit records are found then the content of the target work file remains unchanged.

KEEP — The line numbers of the hit records are retained on copying. If `KEEP` is not specified, EDT creates the target work file as of the current line number which is increased by the current increment for each copied hit record.

OLD The content of the target work file is not deleted before the copy operation. Any existing lines which have the same line number in the work file are overwritten. If OLD is not specified and hit records are found then the target work file is deleted in full before the copy operation (see @DELETE, format 2).

If the specified work file is empty or has been completely deleted and has the character set *NONE then it is assigned the character set of the current work file when the copy operation is performed.

If the specified work file has a character set then the lines that are to be copied are converted into this work file's character set. If characters which cannot be displayed in the work file's character set are identified then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @ ON statement is rejected and error message EDT5453 is output.

When lines are copied to the target work file, record marks are not taken over.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
   2.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   3.00 DUCK     DONALD    WALTSTR.8      DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································································
:                                                                             :
:                                                                             :
:                                                                             :
on 1-5 find mark 2 copy to (3) keep ; 3·····················0001.00:00001(01)
```

Line range 1 to 5 is to be checked for record mark 2 and the hit records are to be copied together with their line numbers to work file 3. Processing then branches to work file 3.

```
   3.00 DUCK     DONALD    WALTSTR.8      DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00
:                                                                             :
:                                                                             :
:                                                                             :
·····························································0003.00:00001(03)
```

9.78 @ON (format 5) – Copy lines with search term

This format of the @ON statement is used to copy all the hit lines in the line ranges that are to be searched into the specified work file.

Operation	Operands	F mode, L mode
@ON	lines[,...] [:cols[:]] **F**IND [**A**LL] [**F**] [**R**] [**NOT**] [**P**ATTERN] search [,int] [**C**OPY [**T**O]] (procnr) [**K**EEP] [**O**LD]	

lines — One or more line ranges in which the search is to be performed.

cols — Contiguous column range to which the search is to be limited.

If the range specification contains only a single column specification, this indicates the range from the specified column through to the end of the line. If the first column specification is greater than the line length then the line is ignored.

If no column range is specified then the column range specified with @SEARCH-OPTION is used.

ALL — Although the specification is permitted, it is pointless since a record can only be copied once.

F — Only the first hit line in each specified line range is copied. If `F` is not specified then all the hit lines from each specified line range are copied.

R — This specification is pointless since whether or not hits are found in a line is independent of the direction of the search.

NOT — A hit is identified if the search term is not present in the specified column range in a line (negative search).

PATTERN — The wildcards present in the search term are interpreted.

search — Search term that is to be searched for in the search range (for details, see section “Searching with @ON” on page 78). It is not permissible to specify an empty string.

int — Only the `int`th occurrence of the search term in a line is considered to represent the first hit. Values between 1 and 32768 are permitted for `int`. The default value for `int` is 1 byte.

procnr — The number of the work file (0..22) to which the hit lines are to be copied.

An active work file or the current work file may not be specified. If OLD is not specified and hit records are found then the target work file is deleted in full before the copy operation (see @DELETE, format 2). If no hit records are found then the content of the target work file remains unchanged.

KEEP — The line numbers of the hit records are retained on copying. If KEEP is not specified, EDT creates the target work file as of the current line number which is increased by the current increment for each copied hit record.

OLD — The content of the target work file is not deleted before the copy operation. Any existing lines which have the same line number in the work file are overwritten. If OLD is not specified and hit records are found then the target work file is deleted in full before the copy operation (see @DELETE, format 2).

If the specified work file has a character set then the lines that are to be copied are converted into this work file's character set. If characters which cannot be displayed in the work file's character set are identified then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @ ON statement is rejected and error message EDT5453 is output.

If the statement is interrupted with [K2] and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·······················
   2.00 HOFER    MARIA     GANGGASSE 3A   80123 MUENCHEN<·······················
   3.00 DUCK     DONALD    WALTSTR.8      DISNEYLAND<···························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·······················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·······················
   6.00 ························································

on & find 'STR.' copy to (5) ; 5·····························0001.00:00001(01)
```

Records which contain the search term STR are to be copied into work file 5. Processing then branches to work file 5.

```
   1.00 DUCK     DONALD    WALTSTR.8      DISNEYLAND<·····························
   2.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·························
   3.00 ···································································
```

```
1·····························································0001.00:00001(05)
```

Processing branches to work file 1.

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·························
   2.00 HOFER    MARIA     GANGGASSE 3A   80123 MUENCHEN<·························
   3.00 DUCK     DONALD    WALTSTR.8      DISNEYLAND<·····························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·························
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·························
   6.00 ···································································
```

```
on & :10-20: find pattern 'M*A' copy to (6) ; 6···············0001.00:00001(01)
```

All the records corresponding to individuals whose first names start with `M` and end with `A` are copied to work file 6. Processing then branches to work file 6.

```
   1.00 HOFER    MARIA     GANGGASSE 3A   80123 MUENCHEN<·························
   2.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·························
   3.00 ···································································
```

```
·····························································0001.00:00001(06)
```

9.79 @ON (format 6) – Replace hit string

If a hit is found then this format of the @ON statement replaces the hit string with a specified string.

Operation	Operands	F mode, L mode
@ON	{ lines / svars } [,...] [:cols[:]] **C**HANGE [**A**LL] [**F**] [**R**] [**P**ATTERN] search [,int] [**T**O] string [**V**]	

lines — One or more line ranges in which the search is to be performed.

svars — One or more ranges of string variables in which the search is to be performed.

cols — Contiguous column range to which the search is to be limited.

If the range specification contains only a single column specification, this indicates the range from the specified column through to the end of the line. If the first column specification is greater than the line length then the line or string variable is ignored.

If no column range is specified then the column range specified with @SEARCH-OPTION is used.

ALL — All the hit strings in a line are replaced by the specified string. If `ALL` is not specified then only the first hit string is replaced.

F — In each specified line range, strings are only replaced in the first hit line. If `F` is not specified then the replacement is made in every hit line of every specified line range.

R — The lines are searched through from right to left. If `R` is not specified then they are searched through from left to right.

PATTERN — The wildcards present in the search term are interpreted.

search — Search term that is to be searched for in the search range (for details, see section “Searching with @ON” on page 78). It is not permissible to specify an empty string.

int — Only the `int`th occurrence of the search term in a line is considered to represent the first hit. Values between 1 and 32768 are permitted for `int`. The default value for `int` is 1 byte.

string — String that is to be replaced by the hit string. It is also permissible to specify an empty string.

The string is converted into the character set used by the work file or string variable. If the string contains characters which cannot be displayed in the target character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @ON statement is rejected and error message EDT5453 is output.

V — The number of hit lines is stored in the integer variable #I2 and the total number of hits is stored in the integer variable #I3. The count is performed irrespective of whether any replacement is performed or not (due to excessive length). The V operand only has an effect if the ALL operand is also specified. If V is not specified then the integer variables #I2 and #I3 remain unchanged.

If the replacement would cause a record to exceed the maximum record length of 32768 characters then no replacement is made, message EDT1937 is output and processing is continued.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   1.00 ABCDEFGHIJKLMNOPQRSTUVWXYZ<·····································
   2.00 WHO ISIS AS IBBORN AS A MULE ?<···································
   3.00 ···································································
on 2 change 'I',3 to 1:19-21 ; on 2 change 'IS' to '' ·····0001.00:00001(01)
```

The first @ON searches for the third occurrence of the character I in line 2. This I is then to be replaced by the string present in columns 19 to 21 of line 1, i.e. STU.

The second @ON replaces the first occurrence of the string IS in line 2 with an empty string, i.e. it is deleted (alternative to @ON format 9).

```
      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   1.00 ABCDEFGHIJKLMNOPQRSTUVWXYZ<·····································
   2.00 WHO IS AS STUBBORN AS A MULE ?<···································
   3.00 ···································································
```

9.80 @ON (format 7) – Replace or insert before or after the hit string

This format of the @ON statement can be used to insert or replace text before or after a hit string in work file lines or string variables.

Operation	Operands	F mode, L mode
@ON	{ lines / svars } [,...] [:cols[:]] **F**IND [**A**LL] [**F**] [**R**] [**P**ATTERN] search [,int] { **C**HANGE / **I**NSERT } { **P**REFIX / **S**UFFIX } string	

lines — One or more line ranges in which the search is to be performed.

svars — One or more ranges of string variables in which the search is to be performed.

cols — Contiguous column range to which the search is to be limited.

If the range specification contains only a single column specification, this indicates the range from the specified column through to the end of the line. If the first column specification is greater than the line length then the line or string variable is ignored.

If no column range is specified then the column range specified with @SEARCH-OPTION is used.

ALL — In the case of a text insertion operation, EDT inserts the string `string` before or after all the hit strings in a line. To identify any further hits, the search continues after the inserted text in the case of a left-to-right search and before the inserted text in the case of a right-to-left search.

In the case of a text replacement operation, EDT replaces the line content with the string `string` before or after all the hit strings in a line. To identify any further hits, the search is continued before or after the replaced text.

If, in the case of a left-to-right search, the text is replaced after a hit string then the specification `ALL` is of no significance. The same applies if text is replaced before a hit string during a right-to-left search.

If `ALL` is not specified then text is only inserted or replaced before or after the first hit string in a line.

F — In each specified line range, text is only replaced or inserted in the first hit line. If F is not specified then the replacement or insertion is performed in every hit line of every specified line range.

R — The lines are searched through from right to left. If R is not specified then they are searched through from left to right.

PATTERN — The wildcards present in the search term are interpreted.

search — Search term that is to be searched for in the search range (for details, see section "Searching with @ON" on page 78). It is not permissible to specify an empty string.

int — Only the intth occurrence of the search term in a line is considered to represent the first hit. Values between 1 and 32768 are permitted for int. The default value for int is 1 byte.

CHANGE — The text located before or after the hit string up to the start or end of the record respectively is replaced by the string specified with string.

INSERT — The string specified with string is inserted in front of or after the hit string.

PREFIX — The string specified with string replaces the line content before the hit string or is inserted in front of the hit string.

SUFFIX — The string specified with string replaces the line content after the hit string or is inserted behind the hit string.

string — String which is to replace the text before or after the hit string or which is to be inserted before or after the hit string. It is also permissible to specify an empty string.

The string is converted into the character set used by the work file or string variable. If the string contains characters which cannot be displayed in the target character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @ON statement is rejected and error message EDT5493 is output.

The string string should be separated from the operands PREFIX or SUFFIX by a blank.

No text is replaced or inserted if this would cause the record to exceed the maximum record length of 32768 characters. Instead, in this case, the message EDT1937 is output and execution is continued.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example 1

```
  23.00 ·····························································
fstat '$user2.xmpl.'·····.....·····0001.00:00001(01)
```

All shareable files under the user ID USER2 which 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 'XMPL.' insert prefix '@READ ''$USER2.'·····0001.00:00001(09)
```

Every partially qualified name XMPL. is to be preceded by the string @READ '$USER2..

```
   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 ····························································· .....




on & find pattern 'XMPL./' insert suffix ''''·····0001.00:00001(09)
```

The closing single quote is inserted after the four file names $USER2.XMPL.1 to $USER2.XMPL.4.

```
   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·····0001.00:00001(09)
```

The four files XMPL.1 to XMPL.4 are read in sequence into work file 1.

Example 2

```
      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   1.00 A11B11C11D11E11F11G11H11<·····································
   2.00 A1111B1111C1111D1111E1111<·····································
   3.00 ·····························································



on 1-2 find R '11',3 insert suffix '++++'·····················0001.00:00001(01)
```

The third occurrence of the string 11 (reading right to left) is to be searched for in the line range 1 to 2 and, if a hit is found, ++++ is to be inserted after it.

```
      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   1.00 A11B11C11D11E11F11++++G11H11<·····································
   2.00 A1111B1111C1111D1111++++E1111<·····································
   3.00 ·····························································



on &:4 find '111',3 change suffix '####'·····················0001.00:00001(01)
```

In line 1, the search term occurred the third time (reading from the right) in columns 17-18.

In line 2, the search term occurred for the first time in columns 24-25, for the second time in columns 22-23 and for the third time in columns 19-20. The string ++++ has been inserted after the hit.

It is now necessary to search through the entire work file as of column 4 for the third occurrence of the string 111. If a hit is found then the text that follows it is to be replaced by ####.

```
   1.00 A11B11C11D11E11F11++++G11H11<·····································
   2.00 A1111B1111C111D111####<·····································
   3.00 ·····························································
```

The search term was not found in line 1.

In line 2, the search term occurred as a hit as of line 4 for the first time in columns 7-9, for the second time in columns 12-14 and for the third time in columns 17-19. The remainder of the line after the hit has been replaced by the string ####.

9.81 @ON (format 8) – Delete hit string

If a hit is found then this format of the @ON statement deletes the hit string when the line content or string variable is being searched through. In this case, the remaining content of the work file lines or string variables is retained.

Operation	Operands	F mode, L mode
@ON	{ lines / svars } [,...] [:cols[:]] **D**ELETE [**A**LL] [**F**] [**R**] [**P**ATTERN] search [,int]	

lines — One or more line ranges in which the search is to be performed.

svars — One or more ranges of string variables in which the search is to be performed.

cols — Contiguous column range to which the search is to be limited.

If the range specification contains only a single column specification, this indicates the range from the specified column through to the end of the line. If the first column specification is greater than the line length then the line or string variable is ignored.

If no column range is specified then the column range specified with @SEARCH-OPTION is used.

ALL — All the hit strings in a line are deleted. If `ALL` is not specified then only the first hit string in a line is deleted.

F — In each specified line range, hit strings are only deleted in the first line containing hits. If `F` is not specified then the hit strings are deleted in each line in each specified line range.

R — The lines are searched through from right to left. If `R` is not specified then they are searched through from left to right.

PATTERN — The wildcards present in the search term are interpreted.

search — Search term that is to be searched for in the search range (for details, see section “Searching with @ON” on page 78). It is not permissible to specify an empty string.

int — Only the `int`th occurrence of the search term in a line is considered to represent the first hit. Values between 1 and 32768 are permitted for `int`. The default value for `int` is 1 byte.

If the statement is interrupted with [K2] and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
   1.00 XXXYYYZZZ *** XXXYYYZZZ ### XXXYYYZZZ %%%<·····························
   2.00 AAA XXXYYYZZZ BBB XXXYYYZZZ CCC XXXYYYZZZ<·····························
   3.00 ······································································

on & delete f r 'XXXYYYZZZ'····································0001.00:00001(01)
```

The lines in the entire work file are to be searched through from right to left for the string XXXYYYZZZ. The first time the search term occurs, the string is to be deleted and the search terminated.

```
   1.00 XXXYYYZZZ *** XXXYYYZZZ ###  %%%<······································
   2.00 AAA XXXYYYZZZ BBB XXXYYYZZZ CCC XXXYYYZZZ<·····························
   3.00 ······································································

on & delete all 'XXXYYYZZZ'····································0001.00:00001(01)
```

In line 1, the first occurrence of the search term when searching from the right was found starting at column 29.

Next, every occurrence of the search term XXXYYYZZZ is to be deleted throughout the entire work file.

```
   1.00  ***  ###  %%%<··················································
   2.00 AAA  BBB  CCC<···················································
   3.00 ······································································
```

9.82 @ON (format 9) – Delete before or after the hit string

This format of the @ON statement causes EDT to delete the content of a work file line or string variable before or after the hit string if a hit is found.

Operation	Operands	F mode, L mode
@ON	{ lines / svars } [,...] [:cols[:]] **F**IND [**A**LL] [**F**] [**R**] [**P**ATTERN] search [,int] **D**ELETE { **P**REFIX / **S**UFFIX }	

lines — One or more line ranges in which the search is to be performed.

svars — One or more ranges of string variables in which the search is to be performed.

cols — Contiguous column range to which the search is to be limited.

If the range specification contains only a single column specification, this indicates the range from the specified column through to the end of the line. If the first column specification is greater than the line length then the line or string variable is ignored.

If no column range is specified then the column range specified with @SEARCH-OPTION is used.

ALL — Each time a hit string is found in a line, the text before or after the hit string is deleted. If, in the case of a left-to-right search, the text after the hit string is deleted then the specification `ALL` is of no significance. The same applies if the text before the hit string is deleted during a right-to-left search. If `ALL` is not specified then the delete operation only applies to the first hit string in a line.

F — In each specified line range, texts are only deleted before or after the hit strings in the first line containing hits. If `F` is not specified then text is deleted before or after the hit strings in each hit line in each specified line range.

R — The lines are searched through from right to left. If `R` is not specified then they are searched through from left to right.

PATTERN — The wildcards present in the search term are interpreted.

search — Search term that is to be searched for in the search range (for details, see section "Searching with @ON" on page 78). It is not permissible to specify an empty string.

int — Only the `int`th occurrence of the search term in a line is considered to represent the first hit. Values between 1 and 32768 are permitted for `int`. The default value for `int` is 1 byte.

PREFIX — The content of the hit line before the hit string is deleted as far as the start of the record.

SUFFIX — The content of the hit line after the hit string is deleted as far as the end of the record.

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

Example

```
   1.00 ABABAB ABABAB ABABAB ABABAB<·····································
   2.00 ABABABABABABABABABABABAB<·····································
   3.00 ·····························································
```

```
on %.-.$ find 'AB'*3,4 delete prefix·····························0001.00:00001(01)
```

The text preceding the hit is to be deleted before the fourth occurrence of the string `ABABAB` (`'AB'*3,4`) in every line in the entire line range (`%.-.$`).

```
   1.00 ABABAB<·····································
   2.00 ABABAB<·····································
   3.00 ·····························································
```

In line 1, the search term was found for the fourth time as a hit starting at column 22.

In line 2, the search term is found as a hit for the first time starting in column 1, for the second time starting in column 7, for the third time starting in column 13 and for the fourth time starting in column 19. When searching for the second and subsequent hits, EDT starts after a hit string each time.

9.83 @ON (format 10) – Delete lines or string variables which contain the search term

This format of the @ON statement causes EDT to delete the line or the content of a string variable which contains the search term. In this case, string variables are reinitialized with a blank and contain the character set EDF041.

Operation	Operands	F mode, L mode
@ON	{ lines / svars } [,...] [:cols[:]] **F**IND [**A**LL] [**F**] [**R**] [**NOT**] [**P**ATTERN] search [,int] **D**ELETE	

lines — One or more line ranges in which the search is to be performed.

svars — One or more ranges of string variables in which the search is to be performed.

cols — Contiguous column range to which the search is to be limited.

If the range specification contains only a single column specification, this indicates the range from the specified column through to the end of the line. If the first column specification is greater than the line length then the line or string variable is ignored.

If no column range is specified then the column range specified with @SEARCH-OPTION is used.

ALL — Although the specification is permitted, it serves no purpose since a line is always deleted when the first hit string is found.

F — Only the first hit line in each specified line range is deleted. If F is not specified then each line containing the hit string in each line range is deleted.

R — This specification is pointless since whether or not hits are found in a work file line is independent of the direction of the search.

NOT — A line is deleted if the search term is not found in the specified column range in a line (negative search).

PATTERN — The wildcards present in the search term are interpreted.

search Search term that is to be searched for in the search range (for details, see section “Searching with @ON” on page 78). It is not permissible to specify an empty string.

int Only the `int`th occurrence of the search term in a line is considered to represent the first hit. Values between 1 and 32768 are permitted for `int`. The default value for `int` is 1 byte.

If the statement is interrupted with [K2] and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

Note

All the empty lines (records of length 0) in a range can be deleted using the statement @ON lines FIND NOT PATTERN '/' DELETE if `'/'` is the wildcard that stands for precisely one character.

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.......................................0001.00:00001(01)
```

All the lines in the work file that contain three or more occurrences of the string `ABC` are to be deleted.

```
   4.00 1  ABC  2  ABC<......................................................
   5.00 1  ABC<..............................................................
   6.00 1<...................................................................
   7.00 .....................................................................



on & : 7 find 'A' delete......................................0004.00:00001(01)
```

Lines 1, 2 and 3 have been deleted. Next, all the lines in the entire work file in which the character `A` occurs after column 7 are to be deleted.

```
   5.00 1  ABC<·····························································
   6.00 1<··································································
   7.00 ····································································



on & find ' '*2 delete·····································0005.00:00001(01)
```

Line 4 has been deleted. Next, all the lines in the entire work file which contain two consecutive blanks '␣␣' are to be deleted.

```
   6.00 1<··································································
   7.00 ····································································
```

Line 5 has been deleted.

9.84 @OPEN (format 1) – Open and read a file

@OPEN (format 1) is used to open an existing file and read it into the current work file or to create a file and open it for processing.

The work file must be empty and a SAM file, ISAM file or library element must not already be open in another work file. The file remains open until it is closed with @CLOSE.

Whenever this section refers to a “file”, this can be a SAM file, an ISAM file, a library element or a POSIX file.

Operation	Operands	F mode, L mode
@OPEN	**L**IBRARY=path1 ([**E**LEMENT=] elname [(vers)] [,eltype]) **E**LEMENT=elname [(vers)] [,eltype] **F**ILE=path2 | *linkname [,**T**YPE=**I**SAM | **S**AM | **C**ATA-] [,**K**EY=**L**INENUMBER | **D**ATA | **I**GNORE] **P**OSIX-FILE=xpath [,**C**ODE=name] [,**M**ODE=**A**NY | UPDATE | NEW | REPLACE]	

LIBRARY= A library element is to be opened and read in. This is defined by explicitly specifying the library name and the element designation.

- path1 Name of the library.
- elname Name of the element.
- vers Version of the required element (see the LMS User Guide [14]). If `vers` is not specified or if `*STD` is specified then the highest available version of the element is selected.
- eltype Type of element. Permitted type specifications are `S`,`M`, `P`, `J`, `D`, `X`, `*STD` as well as freely selectable type names having one of these types as basic type. If `eltype` is not specified then the default type specified with @PAR ELEMENT-TYPE is used. The permitted element types and their meanings are described in chapter “File processing” on page 131.

ELEMENT=... A library element is to be opened and read in. This is defined by means of the element designation without any library name specification. The default library set with @PAR LIBRARY is used implicitly (if @PAR LIBRARY has been specified, otherwise the error message EDT5181 is issued).

The operands elname, vers and eltype have the same meaning as when a library is specified explicitly (see above).

FILE= A BS2000 file is to be opened and read in.

path2 Name of the BS2000 file (fully qualified file name) that is to be opened.

*linkname File link name of the BS2000 file that is to be opened and read in. The file name and the file attributes are stored in the Task File Table. In this way, it is possible to create files with nonstandard names. The file link name must not be specified as the special file name *BY-PROGRAM. This results in the error EDT4923. If no file link name is defined then the statement is rejected with the message EDT5480.

If the file link name is declared as the special file name *DUMMY then it is treated as a nonexistent file. However, no file is created.

TYPE= Specifies the access method for the BS2000 file.

SAM If the file does not yet exist then a SAM file is created, otherwise the specification is ignored. This is the default value when a new file is created.

ISAM If the file does not yet exist then an ISAM file is created, otherwise the specification is ignored.

CATALOG If the file already exists then the attributes are taken over from the category entry, otherwise the message EDT5281 is output. The access method is determined on the basis of the FCBTYPE attribute in the catalog entry. This is the default value for existing files.

KEY= In the case of ISAM files, specifies the location at which the ISAM key is stored in the work file. In the case of other file types, this operand is ignored.

LINENUMBER
The ISAM key is stored as a line number in the work file. This is the default value. If the ISAM key cannot be interpreted as a line number because the position of the key differs from the default value, the key is too long or the keys are not numerical then the message EDT5459 is output and the file is not opened.

DATA The ISAM key becomes a component of the data range in the work file.

IGNORE The ISAM key is not stored in the work file. If the position of the key differs from the default value, the message EDT5466 is output and the file is not read in.

POSIX-FILE= A POSIX file is to be opened.

xpath Path name of the POSIX file that is to be opened.

The `xpath` operand can also be specified as a string variable. It must be specified as a string variable if the path name contains characters which have a special meaning in EDT syntax (e.g. blanks, semicolons in F mode or commas).

CODE= Defines the character set that is to be assumed for the POSIX file. Since it is not possible to assign character sets to POSIX files in the POSIX file system, a user specification is required here.

If `CODE` is not specified then the character set defined in @PAR CODE is assumed.

name Character set of the POSIX file that is to be read in. The name of a valid character set must be specified for `name` (see section “Character sets” on page 47).

EBCDIC The keyword `EBCDIC` is now only supported for reasons of compatibility and is a synonym for the character set `EDF041`.

ISO The keyword `ISO` is now only supported for reasons of compatibility and is a synonym for the character set `ISO88591`.

MODE= Specifies whether the file should already be present.

ANY If the file already exists then it is opened for processing and read in. Otherwise, it is created and opened for processing. This is the default value.

UPDATE The file that is to be opened and read in for processing must already exist or must be linked to `*DUMMY` via the file link name. Otherwise, the message `EDT5281`, `EDT5284` or `EDT5310` is output depending on the file type.

NEW The file is created and opened for processing. It must not already be present as otherwise the message `EDT5258`, `EDT5273` or `EDT5311` is output depending on the file type.

REPLACE If the file already exists then it is opened for processing. However, its previous content is deleted and is not read into the work file. Otherwise, it is created and opened for processing.

If the current work file is not empty or a file is already open in it then the statement is rejected with the message `EDT5191` or `EDT5180`.

If the specified file does not exist or cannot be accessed as required or if an existing file cannot be read in successfully then the statement is rejected with a corresponding error message.

When ISAM files are read with the operand KEY=LINENUMBER, the line numbers in the work file are derived from the file's ISAM key. In all other cases, they are formed using the procedure “insert between two lines” (see section “Line number assignment” on page 36).

If the empty work file has the character set *NONE when an existing file is read in then the work file is assigned the character set of the file that is to be read in. If this character set is *NONE then the work file is assigned the character set EDF003IRV.

If the empty work file already has a character set when an existing file is opened (e.g. due to a preceding @CODENAME statement) then the records that are to be read in are converted from the file's character set into the work file's character set. If the file that is to be read in contains characters which cannot be displayed in the work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the file is not read in and the error message EDT5453 is output.

If the file is present in a Unicode character set and contains an illegal byte sequence, e.g. surrogate characters, then it will be impossible to read it even if SUBSTITUTION-CHARACTERS is specified. In this case, the read operation is rejected with the message EDT5454.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
@OPEN LIBRARY=PROGLIB(ELEMENT=TEST)
```

The element TEST in the library PROGLIB is opened and read into the current work file. In this case, the highest available version and the default type specified with @PAR ELEMENT-TYPE are used. The work file must first be empty.

```
@PAR LIBRARY=LIB1
@SET #S01='PROC.EX'
@OPEN ELEMENT=.#S01(V01),J
```

The element with the name PROC.EX, the version V01 and the element type J (procedure) from the library LIB1 is opened and read into the current work file.

```
@OPEN FILE=FILE1,TYPE=ISAM,MODE=NEW
```

The ISAM file FILE1 is created and opened. The line numbers in the work file represent the ISAM key.

```
@OPEN POSIX-FILE=/home/user1/test/data,CODE=UTF8
```

The POSIX file data in the directory /home/user1/test with the character set UTF8 is opened and read into the current work file.

9.85 @OPEN (format 2) – Real processing of an ISAM file

@OPEN (format 2) opens an ISAM file for processing directly on the disk. This file may already exist, may be created before being opened or be created as a copy of an existing SAM or ISAM file.

Only the section of the ISAM file that is actually required is read at any given time into the current work file. In F mode, these are the records which are fully or partially displayed on the screen. In L mode, the required records are not read in until an EDT statement is executed. The file remains open until processing is terminated by @CLOSE.

It is only possible to open ISAM files for real processing in work file 0. This must be empty or contain a file opened for real processing using @OPEN (format 2).

Operation	Operands	F mode, L mode
@OPEN	[file1] [(ver)] [[**K**EY] [**A**S file2 [[,]**O**VERWRITE]]	

file1 Name of the file that is to be opened or copied. The name must correspond to the SDF data type <filename 1..54>.

Here, the symbolic name '/' for a file for which the file link name EDTSAM or EDTISAM has been assigned by means of the SET-FILE-LINK command is not permitted.

If no file with this name exists and the AS operand is not present then an ISAM file is created and opened. If the AS operand is present then the statement is rejected with the message EDT4971.

If the file is a SAM file and the operand AS is not present then the message EDT4934 is output.

If the file1 operand is not present then the file that has been preset globally with @FILE is opened or copied. If there is no such default setting then the @OPEN statement is rejected with the message EDT5484.

ver Version number of the file. If an incorrect version number is specified for an existing file then the statement is rejected with EDT4985. If the file that is to be opened does not exist then this specification is ignored.

KEY Only effective if a SAM file is copied to an ISAM file which is then opened for real processing. The first 8 characters of each record in the SAM file then form the key of the ISAM file and are to be interpreted as a line number and not as a record content after being read in. This type of record can be created by specifying @WRITE together with the KEY operand.

If file1 is an ISAM file then the KEY operand is ignored.

AS ... The file file1 is copied to an ISAM file. If the copy operation is successful then the copy is opened for real processing.

file2 Name of the ISAM file to which file1 is copied and which is then opened. Here, the symbolic name '/' for a file for which the file link name EDTSAM or EDTISAM has been assigned by means of the SET-FILE-LINK command is not permitted.
If no file with this name exists then an ISAM file is created before the copy operation.

If the file file2 already exists and the operand OVERWRITE has not been specified then the following query is issued in interactive mode:

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

The file file2 is only overwritten and opened if the user responds Y. In batch mode, the file is overwritten and opened.

The file names file1 and file2 must not be identical as otherwise the statement is rejected with the message EDT5489.

OVERWRITE
Suppresses the query EDT0296 if the file file2 exists. The file is overwritten and opened. If file2 does not yet exist then OVERWRITE has no effect.

After @OPEN (format 2), the statements @RENUMBER, @SORT and @COMPARE are rejected. Records from files which have been opened for real processing cannot be marked (mark statement code, @ON, format 3) and are not automatically saved (see @AUTOSAVE).

If a file is already open for real processing then this is implicitly closed when another @OPEN (format 2) statement is issued and the work file is deleted. Only then is the second file opened.

The line numbers in the work file are formed from the file's ISAM key when the file is read. When a SAM file is copied to an ISAM file, the ISAM keys are formed from the first 8 characters of the file if the KEY operand has been specified. Otherwise they are formed using the procedure “Insertion at the current line number” (see section “Line number assignment” on page 36).

The work file is assigned the character set of the file that is to be read in or (if this does not yet exist) the character set defined by the current system settings. If this character set is *NONE then the work file is assigned the character set EDF03IRV. The character set of the work file can then not be changed until the file is closed.

If the work file already has a different character set then the statement is rejected with the message EDT5452.

If the file link name EDTMAIN is assigned to the ISAM file that is to be opened and if the file has been newly created then the attributes stored in the TFT are taken into account when the file is created. Otherwise, a file with default attributes is created. This does not apply if an existing ISAM file is copied and then opened. Such a file is assigned the attributes of the file that is to be copied.

Errors in the file (e.g. non-numerical keys, illegal byte sequences etc.) may sometimes not be detected until much later when the corresponding records are to be read. Not all the records in the file are read in when the file is opened. Any errors that are subsequently detected are then reported by the command which was executing when they occurred. In such cases, a file that has been opened for real processing is automatically closed.

If the ISAM file opened for real processing contains records with identical key values (*duplicate keys*) then it is not possible to predict how EDT will behave in all cases. Depending on the type of read operation, it is possible that either the first or the last of the records with the identical keys will be displayed. In such cases, the first record is always overwritten. If EDT identifies the occurrence of such identical key values then it terminates file processing in the same way as in the presence of other file errors, issues the message EDT5445 and closes the file that has been opened for real processing.

In an ISAM file with a short key (fewer than 8 characters) which has been opened for real processing, it is not possible to create records with line numbers that are too large to be correctly displayed as a key. Statements which cause this are aborted with the error message EDT5446.

If the AS operand is specified and the first file is an ISAM file then it is copied by means of an implicit COPY-FILE command. If it is a SAM file then the copy operation is implemented by means of an implicit @READ statement and an implicit @SAVE statement.

In this case, the statement can be interrupted with K2. If it is interrupted and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Note

This command is suited for processing statements which are too large to be fully loaded into EDT for processing.

9.86 @P-KEYS – Define programmable keys

The @P-KEYS statement is used in interactive mode to load the keyboard's programmable keys (P keys) with a default assignment predefined by EDT or display the EDT predefined default assignment.

Operation	Operands	F mode, L mode
@P-KEYS	[**S**HOW]	

SHOW — This operand displays the predefined EDT default assignment which is loaded into the P keys with the @P-KEYS statement.

If SHOW is not specified then the P keys are loaded with the default assignment predefined by EDT.

In batch mode, the @P-KEYS statement is ignored.

When EDT is started, any existing P key assignment remains unchanged. Similarly, the current assignment is retained when EDT is terminated.

Once the @P-KEYS statement has been executed, the P keys are assigned statements which cause the following actions (output from the @P-KEYS SHOW statement):

```
***  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 P keys have already been loaded with the default EDT settings when the @PAR SPLIT statement is used to modify the division of the screen or the @VDT statement is used to change the format at a 9763 terminal then the @P-KEYS statement must be entered again if the user wants to continue working with the standard EDT assignment for the P keys (this is necessary because the statement sequences stored in the P keys refer to absolute positions on the screen).

If @P-KEYS is entered at a terminal without P keys then the statement is rejected with the error message `EDT5366`.

9.87 @PAGE – Form feed

The @PAGE statement causes a form feed at SYSLST.

Operation	Operands	F mode, L mode
@PAGE		

In addition to the output of the feed control character for a form feed, the maximum number of lines to be printed per page, which is set to a value between 1 and 256 using the @LIST statement, is reset to the default value 65.

9.88 @PAR – Define EDT parameter settings

The @PAR statement is used to define the EDT parameter settings. These settings control the screen display, behavior on input, default values for statements and the declaration of special meanings for certain characters.

Operation	Operands	F mode, L mode
@PAR	[{ $0..$22 | **GL**OBAL }] [[,] parameter[,...]]	

The `parameter` operand consists of a keyword and the associated assigned value. The following syntax description for the possible values of `parameter` is organized thematically. The operands in the operand description are ordered alphabetically.

Parameters for the display and output of the work windows in F mode	
parameter	**E**DIT [-] **F**ULL [={**ON**] | **O**FF}] **E**DIT [[-] **L**ONG] [={**ON**] | **O**FF}] **H**EX [={**ON**] | **O**FF}] **I**NDEX [={**ON**] | **O**FF}] **INF**ORMATION [={**ON**] | **O**FF}] **OPT**IMIZE [={**ON**] | **O**FF}] **PROT**ECTION [={**ON**] | **O**FF}] **SC**ALE [={**ON**] | **O**FF}] **SPLIT** = {n $0..$22 | n (0..22) | **O**FF}

Control of behavior on input and data acquisition	
parameter	**D**ATA-REPLACEMENT [={**ON**] | **O**FF}] **INC**REMENT = {inc | ***S**TD} **LIM**IT = {col | ***S**TD} **LOW**ER [={**ON**] | **O**FF}] **REN**UMBER [={**ON**] | **O**FF}]

Default settings for other statements	
parameter	**C**ODE = {name | **E**BCDIC | **I**SO} [**ELEMENT** [-]] **TYP**E = {eltype | ***S**TD} **L**IBRARY = {path | ***N**ONE} **SDF-NAME**-TYPE = {**I**NTERNAL | **E**XTERNAL | ***S**TD} **SDF-PROG**RAM = {progname | ***N**ONE}

Declaration of special meanings for characters	
parameter	**ESC**APE-CHARACTER = {strspec | ***N**ONE} **SEP**ARATOR = {strchar | ***N**ONE} **STRUC**TURE = {strchar | ***S**TD} **SUBST**ITUTION-CHARACTER = {strchar | ***N**ONE}

$0..$22 The work file to which the @PAR statement refers.
The comma after the work file specification may only be omitted if no further operands are specified.

The operands OPTIMIZE and SUBSTITUTION-CHARACTER always apply to all the work files whereas the SPLIT operand always applies to the current work file. Any work file specification is ignored in these cases. In the case of SPLIT, the warning EDT3127 is also output.

If neither $0..$22 nor GLOBAL is specified then the statement applies to the current work file (if neither OPTIMIZE nor SUBSTITUTION-CHARACTER has been specified).

GLOBAL Specifies that the @PAR statement applies to all the work files. The comma after GLOBAL may only be omitted if no further operands are specified.

The SPLIT operand always applies to the current work file (even if GLOBAL is specified). In this case, the specification of GLOBAL is ignored.

If neither $0..$22 nor GLOBAL is specified then the statement applies to the current work file (if neither OPTIMIZE nor SUBSTITUTION-CHARACTER has been specified).

CODE= Specifies the default value of the CODE operand for statements which read or write POSIX files (see section "POSIX files" on page 134 and the note at the end of the section).

When EDT starts, the value EDF041 is set as the default.

name Name of a valid character set. This is used for the CODE operand in statements which read or write POSIX files unless a character set is explicitly specified for them

EBCDIC This operand is now only supported for reasons of compatibility. It is synonymous with EDF041.

ISO This operand is now only supported for reasons of compatibility. It is synonymous with ISO88591.

DATA-REPLACEMENT=
Specifies whether or not the substitute representation for Unicode characters (see the operand ESCAPE-CHARACTER) is to be taken into account when data is input into the specified work file (in the F mode data window or in L mode) (see section "Substitute character representation in Unicode" on page 52).

When EDT starts, the value DATA-REPLACEMENT=OFF is set as the default.

ON When data is input in the specified work file, substitute representations of the form %uxxxx (where % is the escape character defined with @PAR ESCAPE-CHARACTER and the xs are hexadecimal figures) are replaced by the corresponding Unicode characters. This applies when lines are entered in F mode or in L mode but not, for example, when records are read from files or library elements.

OFF Substitute representations of Unicode characters are only taken into account in literals in EDT statements.

EDIT-FULL= Determines whether both the data window and statement code column are to be set to overwritable in F mode. This setting is only effective if the line number display is active (@PAR INDEX=ON).

When EDT starts, the value EDIT-FULL=OFF is set as the default.

ON The data window and the statement code column are both set to overwritable. It is possible to enter a statement code in a line and at the same time to modify data in this line. This means that it is possible to enter statement codes for data lines that have not yet been created, for example the statement code 0.

Switching to EDIT LONG mode (@PAR EDIT-LONG=ON) or deactivating the line number display (@PAR INDEX=OFF) invalidates this setting. However, the setting is not deactivated. Instead, it is recorded and becomes effective again as soon as the conflicting mode is deactivated.

The specification of @PAR EDIT-FULL=ON is ignored as long as write protection (@PAR PROTECTION=ON) is active.

OFF In a screen line, it is only possible to write the statement code column or the data section.

EDIT-LONG= Determines whether records that are longer than a screen line are to be displayed in full or truncated in the work window in F mode.

When EDT starts, the value EDIT-LONG=OFF is set as the default.

ON If possible, records are displayed in full in the work window. In EDIT-LONG mode, neither the column counter activated with @PAR SCALE=ON nor an information line requested with @PAR INFORMATION=ON are displayed. The column counter and information lines are not displayed until EDIT-LONG mode is exited. For details of the screen layout in EDIT-LONG mode, see section "The work window" on page 103.

Activating EDIT-LONG mode implicitly deactivates the line number display (@PAR INDEX=OFF) and hexadecimal mode (@PAR HEX=OFF).

OFF If necessary, records are displayed in truncated form. The length of the displayed section depends on the terminal and the setting made with @VDT or @PAR INDEX: 72, 80, 124 or 132 characters per screen line.

The line number display remains active when EDIT-LONG mode is exited. EDIT-LONG mode is also deactivated by @PAR INDEX=ON and @PAR HEX=ON.

ELEMENT-TYPE=

Specifies the default element type for library elements.

This type of element is accessed if no element type is specified in the @COPY, @OPEN, @DELETE, @WRITE or @INPUT statements (see the note at the end of the section).

When EDT starts, the value S is set as the default.

eltype Permitted type specifications are S, M, P, J, D, X, R, C, H, L, U, F and freely selectable type names having one of these types as basic type. The eltype operand can also be specified as a string variable in the form ELEMENT-TYPE=.svarex. The period must be specified in order to make it possible to distinguish the name of the string variable from a freely selectable type name starting with #.

The non-text types (R, C, H, L, U, F) are not permitted in statements used for reading and writing. Specifying them in @PAR ELEMENT-TYPE is therefore only of any use if the default setting is used only for the @DELETE statement. The permitted element types and their meanings are described in section "File processing" on page 131.

*STD Restores the default value (i.e. the value S).

ESCAPE-CHARACTER=

Specifies the character used to introduce the substitute representation for Unicode characters (see also section "Substitute character representation in Unicode" on page 52 and the note at the end of the section).

When EDT starts, the value ESCAPE-CHARACTER=*NONE is set.

strspec Special character that acts as an escape character to introduce a Unicode substitute representation. If, for example, @PAR ESCAPE-CHARACTER='%' has been used to declare % as the escape character then a string of the form %Uxxxx with four hexadecimal digits x is interpreted as the substitute representation for the Unicode character with the code xxxx. The setting for @PAR DATA-REPLACEMENT controls whether the substitute representation is only to be taken into account for literals in statements or is also to apply to data input in the work window or in L mode. The escape character for the current work file is always used as the basis for the interpretation of both statements and data input.

*NONE No escape character is assigned. EDT assumes that the input does not contain any substitute representations.

HEX= This is used to activate and deactivate hexadecimal mode (see section "Hexadecimal mode" on page 120). In F mode, the lines for which hexadecimal mode has been activated are displayed on screen in both printing and hexadecimal form. Details concerning the layout can be found in the section on hexadecimal mode.

When EDT starts, the value HEX=OFF is set.

ON Hexadecimal mode is activated for the specified work file. This implicitly deactivates EDIT-LONG mode.
If the same work file is displayed in multiple work windows on the screen then the same setting is used in both data windows.

If, in the case of split screen display, the work window is so small that it is not possible to display even one data line together with its hex lines then the message EDT2404 is output. Hexadecimal mode is activated nevertheless. The user can then enlarge the data window so that the hex lines can also be displayed.

OFF Hexadecimal mode is deactivated.

INCREMENT= Specifies the increment that is to be used when line numbers are assigned. For a description of the statements and actions to which this setting applies, see section "Line number assignment" on page 36.

When EDT starts, the value `1.0` is set as the default.

inc Increment that is to be used when line numbers are assigned. If @PAR INCREMENT is specified with an increment `<0.01` then it should be noted that the line numbers of read, copied or inserted lines are not displayed in full in F mode (6-digit line number display). Unwanted results may occur if these incompletely displayed line numbers are used in statements.

*STD The default value on EDT start is restored.

INDEX= Activates and deactivates the line number display in F mode. The screen layout with the line number display activated and deactivated is described in section "F mode" on page 101.

The setting for the line number display is saved separately for the upper and lower (possible) data windows corresponding to each work file. If only one data window is displayed on the screen then its settings overwrite those of the upper data window for the corresponding work file.

If the statement @PAR GLOBAL,INDEX=... has been entered then it applies to all the (possible) screen windows corresponding to all the work files. If only a single work file is specified (including the implicit specification of the current work file) then it is necessary to distinguish between a number of different cases:

If the screen is *split* (see @PAR SPLIT) then the @PAR INDEX statement only applies to the currently visible screen window corresponding to the specified work file. If both screen windows corresponding to the specified work file are visible then the statement applies to the entry window.

If neither of the screen windows corresponding to the specified work file are visible then the statement applies to the upper screen window. The equivalent comments apply to input in L mode, where *visibility* and *entry window* refer to the state following any switch to F mode (using the @EDIT FULL statement). The corresponding information can be displayed in L mode using @STATUS=PAR(..).

If the screen is *not split* then the statement applies to both (possible) screen windows of the specified work file irrespective of whether these are visible or not. The equivalent comments again apply in L mode, i.e. if a non-split screen is displayed after @EDIT FULL.

When EDT starts, the value `INDEX=ON` is set as default.

ON — The line number display is activated. Specifying @PAR INDEX=ON also deactivates EDIT-LONG mode.

OFF — The line number display is deactivated. When the line number display is deactivated, EDIT-FULL mode ceases to be effective but is not deactivated.

INFORMATION=

In F mode, specifies whether an information line for the specified work file is to be output in the data window.

Alongside the number of the work file and the name of the character set defined for this work file if one exists, the information line contains the name of a local @FILE entry or the name of the open file or library element (see section "File processing" on page 131). If work file 9 contains the output from one of the statements @FSTAT LONG, @STAJV LONG or @SHOW (without a target specification) and the content has not been changed then the information line in work file 9 contains column headers for the associated output. If none of these cases applies then the information line contains only the number of the work file and the character set.

When EDT starts, the value INFORMATION=OFF is set as default.

ON — The information line is displayed as the first line in the work window ahead even of any column counter that may have been activated. The screen display is not modified if the work window is too small to display at least one data line in addition to the information line. The setting does not take effect until the work window is enlarged accordingly.

OFF — No information line is output.

LIBRARY= — Specifies the default library name for the statements @COPY, @OPEN, @WRITE, @INPUT and @SHOW. This default setting is used if no library name is specified in the above-mentioned statements (see note at the end of the section).

When EDT starts, the value LIBRARY=*NONE is set as default.

path — Name of the library.

*NONE — The default setting for the library name is reset. In the above-mentioned statements, it is then necessary to specify the library explicitly.

LIMIT= Defines the maximum record length in the F mode data window. If a record is input with a length which exceeds the specified value then the record is truncated and the message EDT2267 is output.

Indirect modifications to records due to statements or data entry in L mode are unaffected by this check. In particular, this parameter has nothing to do with the record length check set using the @CHECK or @TABS statements in L mode.

When EDT starts, the value LIMIT=32768 is set as default.

col The maximum permitted record length (1..32768) for input in the F mode data window.

*STD The default value on EDT start is restored.

LOWER= Specifies whether or not EDT converts lowercase characters entered at the terminal into uppercase (see also note at the end of the section).

When EDT starts, the value LOWER=ON is set as the default.

ON EDT differentiates between uppercase and lowercase. Texts and strings are processed in the form that they are entered.

OFF EDT converts lowercase characters entered at the terminal into uppercase. In F mode, any lowercase characters present in the work file are converted into smudge characters for output in the work window.

OPTIMIZE= Activates and deactivates the optimization of screen output. Before each screen output, EDT compares the screen that is to be output with the previous screen. By default, it only outputs the modified text in order to improve the output (optimization). The unchanged text in the old screen is retained unmodified.

When EDT starts, the value OPTIMIZE=ON is set as default.

ON In each dialog step, only the modified line contents are output.

OFF The entire content of the work window is output in each dialog step.

PROTECTION=

Activates or deactivates record-level write protection. This operand is only of use if the EDT subroutine interface is used. Records can then be marked accordingly to display them as write-protected or overwritable from within the user program (see the Subroutine Interfaces User Guide [1]).

When EDT starts, the value `PROTECTION=OFF` is set as default.

ON — Correspondingly marked records are displayed as write-protected or are automatically set to overwritable in the F mode dialog.

If @PAR EDIT-FULL=ON has already been used to set both the statement code column and the data lines to overwritable then this setting is reset.

OFF — The presetting specified by the user program (write-protected or overwritable) does not apply.

RENUMBER= Controls the automatic renumbering of line numbers. For a description of the statements and actions to which this setting applies, see section “Line number assignment” on page 36.

When EDT starts, the value `RENUMBER=ON` is set as default.

ON — The line numbers in a work file are renumbered if required. EDT does this if the increment is not small enough to execute the statement in full when reading a file (or library element) or when performing a copy or insert operation in a file.

OFF — EDT does not modify the line numbers in a file. EDT issues a message if it is not possible to execute the statement because the increment is not small enough to accommodate all the records.

SCALE= In F mode, activates or deactivates the display of a column counter (horizontal ruler) in the data window.

The column counter is not output in `EDIT-LONG` mode. In the case of output in hexadecimal mode, there is not just one column counter. Instead, a column counter is output for every line. However, the setting made with @PAR SCALE becomes effective if the corresponding mode is exited.

When EDT starts, the value `SCALE=OFF` is set as default.

ON — The column counter is displayed as the first line after an information line, if one is present (see @PAR INFORMATION), and displays the current work window line numbers for the required work file (e.g. after the work window has been moved horizontally).

If a tabulator has been defined (see @TABS statement), an additional screen line is displayed in which the current position of the tabulator is displayed with `I`. In this line, the tabulator character itself is depicted in the statement code column.

If the work window is too small to display at least one data line in addition to the column counter then the column counter is not displayed. As the work window is enlarged, the information line (if present) is shown again first, followed by the column counter and then, finally, the tabulator line (if present).

OFF Deactivates the column counter and the tabulator display scale if present.

SDF-NAME-TYPE=

Specifies the name type of the program name which has been predefined with @PAR SDF-PROGRAM as well as the default setting for the name type in the @SDFTEST statement (see also the note at the end of the section).

When EDT starts, the value `SDF-NAME-TYPE=INTERNAL` is set as default.

INTERNAL

The program name is the maximum 8-character internal name. The internal name can be ascertained using `SDF-A` if it does not correspond to the name of the program.

EXTERNAL

The program name is the maximum 30-character external name (e.g. LMS, SDF-A or HSMS).

*STD The default value on EDT start is restored.

SDF-PROGRAM=

Defines the name of a program for the @SDFTEST statement and the statement code `T`. When this statement or statement code are executed then data lines which start with `//` are analyzed using this program's syntax file (see also the note at the end of this section).

When EDT starts, the value `SDF-PROGRAM=*NONE` is set as the default.

progname Name of a program. Either the internal name with a maximum of 8 characters or the external name with a maximum of 30 characters can be defined (see @PAR SDF-NAME-TYPE)

*NONE `*NONE` cancels the program definition.

SEPARATOR= Specifies the record separator used to delimit records or the character which acts as the default value for the @SEPARATE statement (see section "Statement in data window – splitting a record" on page 112 or @SEPARATE statement and the note at the end of the section).

When EDT starts, the value `SEPARATOR=*NONE` is set as the default.

strchar The record separator can be specified as an alphanumeric character or special character in single quotes or in the form `U'unicode'` in its `UTF16` coding.

Different characters must be chosen for the record separator and the tabulator character.

*NONE Cancels a record separator definition. For reasons of compatibility, the keyword `OFF` is also permitted here.

SPLIT= In F mode, `SPLIT` can be used to output a second work window on the screen. The work file in which this statement was entered (current work file) is displayed in the upper work window. The work file specified with `$0..$22` is displayed in the lower work window. When EDT starts, the value `SPLIT=OFF` is set as the default.

n $0..$22 | n (0..22)
The specified work file is displayed with length `n` in the lower work window. A value of `2` or more must be specified for the length `n` and this value must not be more than two less than the maximum number of lines permitted by the terminal for the screen format in question (see @VDT statement).

The file position (line, column number) in the work file specified here is retained when @PAR is issued. In the subsequent EDT dialog, the file positions of the two work files can be modified individually.

OFF Hides the second work file and sets the length for the remaining screen window to the maximum value permitted by the terminal. The screen window containing the statement line in which the statement was issued remains present. If @PAR SPLIT = OFF is entered in the upper work window while statements are present in the lower work window then @PAR SPLIT is rejected with an error message. The display of two work windows is also implicitly deactivated by the @VDT statement.

STRUCTURE=

Specifies the structure symbol that is evaluated when positioning is performed on the basis of the structure depth (see the statement codes `+` and `-`).

When EDT starts, the value `'@'` is set as the default.

strchar Character that is to be used as the structure symbol. This symbol indicates records which have to be taken into account when positioning is performed on the basis of the structure depth. The character can be specified as an alphanumeric character or special character in single quotes or in the form `U'unicode'` in its `UTF16` coding.

If a character other than a blank is defined as the structure symbol then only lines that contain this structure symbol are positioned to when scrolling is performed on the basis of structure depth. If a blank is specified as the structure symbol then it is possible to position to all the lines.

*STD The default value on EDT start is restored.

SUBSTITUTION-CHARACTER=

Specifies the substitute character that is used instead of a character which is invalid in the target character set during conversion. The setting applies for all work files and string variables irrespective of the work file in which the @PAR statement was issued. This setting does not apply when characters are converted into the communications character set (see section “Character sets” on page 47). In this case, the terminal's device-specific smudge character is always used as the substitute character.

When EDT starts, the value `SUBSTITUTION-CHARACTER=*NONE` is set as the default.

strchar Character that is used instead of a character which is invalid in the target character set during conversion. The character can be specified as an alphanumeric character or as a special character in single quotes or in the form `U'unicode'` in its `UTF16` coding.

Since it is not possible to declare a separate substitute character for every possible target character set, it only makes sense to choose characters which are present in every employed character set. For example, a number of the characters in the EBCDIC DF03 kernel (`&-/:.,<*%_+>?`) or the character `X'00'` may be suitable (see section “Character sets” on page 47). However, this is not checked on input. If conversion is to be performed using an invalid character set then EDT behaves as if `*NONE` had been defined for the substitute character.

*NONE On conversion, no substitute representation is used for characters that are invalid in the target character set. Instead, conversions that would result in such characters are rejected with an error message. Outputs to SYSOUT and SYSLST are an exception this rule. In this case, blanks are used.

If no operands are specified then all the presettings for the current work file (with the exception of SPLIT and OPTIMIZE) are reset to the values that apply when EDT is started. The value for SUBSTITUTION-CHARACTER is always reset to this default value for all the work files.

If only the operands GLOBAL or $1..$22 are specified then all the presettings for all the work files or all the presettings for the specified work file (with the exception of SPLIT and OPTIMIZE) are reset to these default values. The value for SUBSTITUTION-CHARACTER is always reset to this default value for all the work files.

The settings operands are set in the order in which they are specified. This is of significance in the case of parameters which affect one another or when a statement is aborted due to a runtime error (may occur in the case of SPLIT and CODE).

The @PAR statement can also be used in L mode to change settings which affect the display in F mode. Although these take effect immediately, this is, of course, only visible, when processing switches to F mode.

With the exception of INDEX, all the settings for a work file apply irrespective of whether one, both or neither of the possible screen windows for this work file are displayed. Thus, if the same work file is displayed on the screen more than once, the setting applies to both work windows (except in the case of INDEX).

The current values for the settings made with @PAR can be output using the @STATUS statement.

Note

The settings that are made with @PAR (except in the case of OPTIMIZE, SPLIT and SUBSTITUTION-CHARACTER) can be set differently for each work file.
This also applies to the settings made with CODE, ELEMENT-TYPE, ESCAPE-CHARACTER, LIBRARY, LOWER, SDF-NAME-TYPE, SDF-PROGRAM and SEPARATOR which affect the default values or the behavior of another statement (statement using the value). In the case of these settings, the rule is that the value of the work file set as the current work file when the statement using the value is active should apply.
If it is permissible to specify string variables for a value then the value is replaced by the content of the string variables when the @PAR statement is executed. Consequently, any later changes to the string variables have no effect on the setting.

9.89 @PARAMS – Define procedure parameters

The @PARAMS statement can be used to define symbolic parameters which are used in @DO procedures. There is a distinction between keyword parameters and positional parameters.

The parameters can be considered as string variables which are replaced, before the EDT procedure is executed, by the corresponding values which were specified when the procedure was called in the @DO statement or, in the case of keyword parameters, are defined as the default values in the @PARAMS statement itself.

Operation	Operands	@PROC
@PARAMS	[formal[,...]] [[,] {formal=param} [,...]]	

formal — Formal (symbolic) parameter. A parameter starts with the character `&`. This is followed by a letter (`A..Z, a..z`) which in turn can be followed by up to 6 letters or digits. Lowercase characters may also be used. It should be noted that EDT differentiates between uppercase and lowercase. The specifications `&A` and `&a` therefore designate different parameters. The parameter names used in a work file only apply within this work file. If an illegal formal parameter is specified then execution of the procedure is rejected with the message `EDT4924`.

param — Default value of the keyword parameter. The specified value is used in the procedure if no other value was specified for this keyword parameter when the procedure was called with the @DO statement. Keyword parameters must be declared after all the positional parameters.

The @PARAMS statement must be located in the first line of a @DO procedure. A @PARAMS statement in continuation lines is ignored and the message `EDT5479` is output. Errors in the @PARAMS statement interfere with the execution of the procedure and are reported by @DO. The @PARAMS statement is not itself a component of the procedure and can, for example, not be branched to, is not logged if the procedure is executed with the setting @DO...PRINT and is not executed more than once in procedures which contain external loops.

At least one formal parameter must be specified. Otherwise execution of the procedure is rejected with the message `EDT4918`. The same message is output if a parameter does not start with `&` or there is no further parameter after a comma.

If positional parameters are specified after keyword parameters then the message `EDT4948` is issued. If a keyword parameter is specified more than once then the error message `EDT3910` is output. In the event of other errors, the message `EDT5478` is output.

The number of formal parameters possible in @PARAMS is only limited by the maximum permitted length of EDT statements.

In the @PARAMS statement, EDT ignores blanks which occur immediately before or after a parameter name. The specification of blanks in the parameter name which was possible in compatibility mode is no longer supported.

In keyword parameters, the parameter name (keyword) is followed by an equals sign. The equals sign is followed by the parameter's default value. Keyword parameters are also assigned the current value from the parameter list in the @DO call. If this value is not present there then the predefined default value specified in @PARAMS is used. The assigned default value consists of all the specified characters, including blanks. If no default value is to be assigned to a keyword parameter then a comma must be specified immediately after the equals sign or @PARAMS must be terminated. Alternatively, the explicit specification of an empty string '' as the default value is also possible.

The default value of a keyword parameter may be enclosed in single quotes. These are not transferred if they occur as the first or last character in the parameter value and no quotes or only double quotes occur between them. In all other cases, all the specified single quotes form part of the parameter (see example). The comma and closing parenthesis characters may only form part of a parameter if they occur in a substring in the parameter value that is enclosed by single quotes. Single quotes must always occur in pairs in a parameter value. An individual single quote cannot be passed in a parameter value. If @QUOTE has already been used to assign the function of the single quote to another character then this does not apply to the single quotes enclosing the default value.

If no value is specified for a positional parameter when the procedure is called with @DO then it is assigned an empty string as its value. If no value is specified for a keyword parameter when the procedure is called with @DO then it is assigned the default value specified in the @PARAMS statement as its value.

The positional parameters must be specified in the same order in @PARAMS and in @DO. The sequence of keyword parameters can be different in @PARAMS and @DO (see the @DO statement).

If the procedure file contains a string which starts with & and could syntactically be identified as a formal parameter which, however, is not declared in the @PARAMS statement or if there is no @PARAMS statement then the corresponding string is not replaced.

The parameters can be used at any location within the procedure and can be linked together with other strings and parameters.

If, in the procedure, the formal parameter is directly followed by a letter, a digit, or a period then the formal parameter must be separated from these characters by a period (except in the case of parameter names with a maximum length of 7).

A single period after a formal parameter is interpreted as a delimiter and is not taken over when the formal parameter is replaced by the corresponding value (see example). This applies independently of the character that follows the period.

If a string which starts with & and corresponds to one of the formal parameters listed in @PARAMS is not to be replaced by the current parameter value in the procedure file then the & character must be doubled. When the procedure executes, one of the two &s is eliminated. Any other doubled &s are reduced to a single & if the procedure contains a @PARAMS statement.

If a formal parameter is replaced by the current parameter value in the procedure then this may cause the line to exceed the maximum permitted line length. This causes the output of error message EDT1938 during the execution of the procedure. The line in question is not executed.

For information on the handling of character sets during parameter processing, see section "EDT procedures" on page 64 and the description of the @DO statement.

The indirect specification of operands is not permitted for this statement.

Example 1

Parameter value specification	Substituted string
&F=A,...	A
&F=,...	Empty string
&F='',...	Empty string
&F=␣ABC␣,...	␣ABC␣
&F='X',...	X
&F='X''X',...	X''X
&F='X'Y'X',...	'X'Y'X'
&F='AB'C,...	'AB'C
&F=␣'ABC',...	␣'ABC'
&F=␣,...	␣
&F=',)',...	,)
&F=A','B,...	A','B

Example 2

Line in procedure file	Parameter input	Generated line
&PARAM(BC)	&PARAM=A	A(BC)
&PARAM.(BC)	&PARAM=A	A(BC)
&PARAM..(BC)	&PARAM=A	A.(BC)
&PARAM..BC	&PARAM=A	A.BC
&PARAM.2BC	&PARAM=A	A2BC
&PARAMBC	&PARAM=A	&PARAMBC
&PARAM,.2B	&PARAM=A	A,.2B
BC&PARAM	&PARAM=A	BCA
BC,&PARAM	&PARAM=A	BC,A
B2&PARAM	&PARAM=A	B2A
&PARAM.&PARAM	&PARAM=A	AA
&PARAM&PARAM	&PARAM=A	AA
&PARAM..&PARAM	&PARAM=A	A.A
&PARAM&&PARAM	&PARAM=A	A&PARAM
@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
&CMD #S1	&CMD=@PRINT	@PRINT #S1

Example 3

```
   7.     @PRINT
   1.0000 THE OUTPUT
   2.0000 FROM THIS
   3.0000 PROCEDURE
   4.0000 IS DETERMINED
   5.0000 IN THE @DO
   6.0000 COMMAND
   7.     @SET #S3 = '*** AS YOU SEE ***'
   7.     @PROC 1
   1.     @ @PARAMS &LINES ------------------------------------------------ (1)
   2.     @ @NOTE Output &LINES
   3.     @ @PRINT &LINES ------------------------------------------------- (2)
   4.     @END
   7.     @DO 1(2-4) ------------------------------------------------------ (3)
   2.0000 FROM THIS
   3.0000 PROCEDURE
   4.0000 IS DETERMINED
   7.     @DO 1(#S3),PRINT
   7.     @NOTE Output #S3 ------------------------------------------------ (4)
   7.     @PRINT #S3
     #S03 *** AS YOU SEE ***
   7.     @DO 1(2,4N) ----------------------------------------------------- (5)
 %  EDT4963 TOO MANY OPERANDS
   7.     @DO 1('2,4N') --------------------------------------------------- (6)
   2.0000 FROM THIS
IS DETERMINED
   7.
```

(1) The positional parameter `&LINES` is declared in the first line of work file 1.

(2) This parameter occurs in the @PRINT statement. Which lines are now output depends on the parameter value specified in the @DO statement.

(3) Work file 1 is executed. However, before the individual statements are executed, the range of values 2-4 is assigned to the parameter `&LINES`.

(4) The use of the parameter values becomes particularly clear if the individual statements in the procedure are output on the screen before they are executed since the parameter values have already been inserted at this point.

(5) If the user attempts, for example, to display line 2 with a line number and line 4 without a line number then a comma which forms part of the parameter value is interpreted as a separator between two parameters and the @DO statement is therefore rejected.

(6) A parameter value can be passed in single quotes. In this case, the value that is located between the single quotes is assigned to the `&LINES` parameter. In this way, it is also possible to transfer commas as a part of the parameter value.

Example 4

```
   1.     @PROC 2
   1.     @ @PARAMS &STRVAR1,&STRVAR2,&CONTENT1=**** -------------------- (1)
   2.     @ @SET &STRVAR1 = '&CONTENT1'
   3.     @ @SET #S2 = '&STRVAR1'
   4.     @ @SET #S3 = &STRVAR1
   5.     @ @SET &STRVAR2 = &STRVAR1
   6.     @ @SET #S4 = 'FROM &STRVAR1 TO &STRVAR2'
   7.     @ @PRINT &STRVAR1,&STRVAR2,#S2,#S3,#S4
   8.     @END
   1.     @DO 2(#S0,#S1) ------------------------------------------------ (2)
     #S00 ****
     #S01 ****
     #S02 #S0
     #S03 ****
     #S04 FROM #S0 TO #S1
   1.     @DO 2(#S15,#S13,CONTENT1=BLABLA) ------------------------------ (3)
     #S15 BLABLA
     #S13 BLABLA
     #S02 #S15
     #S03 BLABLA
     #S04 FROM #S15 TO #S13
1.
```

(1) Two positional parameters and one keyword parameters are defined for work file 2.

(2) The values of the positional parameters in @DO must be specified in a sequence which corresponds to the sequence of the positional parameters in the @PARAMS line. When work file 2 is executed, the value of `&STRVAR1` is set to `#S0` and the value of `&STRVAR2` is set to `#S1`. Since no value is specified for the keyword parameter `&CONTENT1`, the default value, i.e. `****`, is used at runtime.

(3) The specification of a parameter value for a keyword parameter in @DO replaces the default value.

Example 5

```
1.     @PROC 3
1.     @ @PARAMS &A,&B,&C,&X=111,&Y=222,&Z=333 ----------------------- (1)
2.     @ @CREATE #S10: '&A','&B','&C','&X','&Y','&Z'
3.     @ @PRINT #S10
4.     @END
1.     @DO 3 (AAAAA,BB,CCCCCCC) ------------------------------------- (2)
  #S10 AAAAABBCCCCCCC111222333
1.     @DO 3 (AA,BBBB,C,Y=****,X=########) --------------------------- (3)
  #S10 AABBBBC########****333
1.
```

(1) Three positional and three keyword parameters are defined in work file 3.

(2) Work file 3 is executed. However, since no values have been specified for keyword parameters, the default values are used.

(3) Now, values are specified for two of the keyword parameters. It should be noted that the sequence of values specified for the keyword parameters does not correspond to the sequence used for the definition of the keyword parameters in the @PARAMS line.

9.90 @PREFIX – Insert string as prefix

The @PREFIX statement is used to insert a string as a prefix in front of every line or string variable in each specified range (see also @SUFFIX).

Operation	Operands	F mode, L mode
@PREFIX	{ lines / svars } [,...] **W**ITH string	

lines — One or more line ranges in which text is to be inserted at the start of each line. Only existing lines are processed.

svars — One or more ranges of string variables in which text is to be inserted at the start of each string variable.

string — String that is to prefix each line or string variable in each specified range. It is also permissible to specify an empty string.

The string is converted into the character set used by the work file or string variable. If the string contains characters which cannot be displayed in the target character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @ PREFIX statement is rejected and error message EDT5453 is output.

If inserting the string would cause a line or string variable to exceed the maximum record length of 32768 characters then it is not inserted and the message EDT5474 is output.

If errors occur during processing (EDT5453 or EDT5474) then the statement is aborted. Any lines and/or string variables which have been successfully modified up to this point retain their changes.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
   1.00 AND<·····························································
   2.00 ONCE<····························································
   3.00 AGAIN<···························································
   4.00 AGAIN<···························································
   5.00 AGAIN<···························································
   6.00 ·····························································
```

```
prefix 4-5 with ' ONCE '·····································0001.00:00001(00)
```

The string `ONCE` is to be inserted as a prefix in the line range 4-5.

```
   1.00 AND<·····························································
   2.00 ONCE<····························································
   3.00 AGAIN<···························································
   4.00  ONCE AGAIN<·····················································
   5.00  ONCE AGAIN<·····················································
   6.00 ·····························································
```

```
prefix 4-5 with 1·····························································0001.00:00001(00)
```

The content of line 1 is to be inserted as a prefix in the line range 4-5.

```
   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·····································0001.00:00001(00)
```

Five blanks are to be inserted as a prefix in the line range 4 to 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·····································0001.00:00001(00)
```

The content of line 4 is to be inserted as a prefix in the line range 4-5.

```
   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 ·····························································
```

9.91 @PRINT – Print or output line ranges or the content of string variables

The @PRINT statement outputs the content of the specified line ranges or string variables. In interactive mode, the output is written to SYSOUT and in batch mode it is written to SYSLST.

For the sake of simplicity, the operand description refers primarily to lines and line ranges. However, the descriptions apply equally to string variables and ranges of string variables unless explicitly indicated to the contrary.

Operation	Operands	F mode, L mode
@PRINT	[{lines / svars} [:cols[:]] [**X**] [**N**] [**S**] [{**V** / **E**}]] [,...]	

lines — Line range that is to be output.

svars — Range of string variables whose content is to be output.

cols — Column range in the work file or in the string variables that is to be output.

If only one column number is specified then the remainder of the line is output as of this column. If the first column specification is greater than the line length then the line is not output.

If no column range is specified then the entire line or string variable is output even if it is empty.

X — The format of the output is hexadecimal. Between two and eight hexadecimal numbers are output for each character depending on the character set which has been defined for the work file or string variable.

N — Removes the line numbers or string variable names from the output.

S — If the output is sent to SYSLST then the first line of each range is usually output with an additional line feed if the output does not start at the beginning of a page. If S is specified then this empty line is not output. The operand is only meaningful in batch mode and is ignored in interactive mode.

V, E

In interactive mode, the operands V and E cause EDT to output the specified line range one section at a time (depending on the screen size) and possibly also to give the user the opportunity to enter scrolling statements at the end of each section (see below). The number of physical lines (screen lines) present in a section depends on the employed terminal.

These operands are only meaningful when working at the screen and are ignored in batch mode.

If V is specified (or if a scrolling statement causes a switch to V mode) then the user is always prompted to enter a scrolling instruction at the end of each section.
To terminate output or move to the next specified range, the user must enter O or change to E mode (see below).

In contrast, if E is specified (or if the user switches to E mode by means of a scrolling statement) then the user is only prompted to enter a scrolling statement at the end of a section if this section does not itself contain the last line of the current range. If the section that is to be output contains the last line of the current range then the section is output without any scrolling request if it is not followed by any further ranges or if the following range does not make use of either the V or the E operand.

If another range has been specified with the V or E operand then the last section of the current range is joined with the start of the next range to form a single section. The way EDT behaves at the end of this composite section is determined by the V or E operands of the next range.

If neither V nor E is specified then the complete range is output. Output in interactive mode is only interrupted if the operating system's overflow monitoring (/MODIFY-TERMINAL-OPTIONS OVERFLOW-CONTROL=USER-ACKNOWLEDGE) is active.
After an interruption of output via %PLEASE ACKNOWLEDGE, the user can choose K2 and /RESUME-PROGRAM to abort the output of the line range or enter any other value to continue the output. Positioning within the line range is not possible.

If no operand is specified then the entire current work file is output one section at a time (in the same way as if E is specified).

The content of the specified line range is output line-by-line with or without a prefixed line number (see the N operand). If SYSOUT is assigned to a terminal in interactive mode then the individual lines are separated from one another by LZE (logical end-of-line).
It is possible to specify multiple ranges with different output options provided that these are separated by commas. The number of range specifications is only limited by the maximum permitted length of EDT statements.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

In the case of section-by-section output in interactive mode, EDT prompts the user to enter a *scrolling instruction* after outputting a *section* (section of a file or sequence of string variables). This enables the user to interrupt output or move to a required position. In this case, it is also possible to exit the range defined with lines or svars. The following scrolling instructions are possible:

DUE — Void input on scrolling causes EDT to go to the next section of the output (i.e. the section immediately following the last one output). In V mode, if the end of the current range has already been reached then the last line in this range is output again and the user is once more asked to enter a scrolling instruction.
When the output of a range starts, the current range is always equal to the specified range. However, this can be modified using the scrolling statements + or −.

* — Switches to E mode and resets the current range (if it has been changed) to the range specified in the statement.
EDT then moves on to the next section of the output (i.e. the section immediately following the last one that was output) unless the last line that was output is the last line of the specified range or already exceeds the specified range (possible if + has been entered). In this case, EDT moves on to the next specified range or terminates output.

\+ — Switches to V mode and the range is set to 0.0001-9999.9999 for lines and #S01-#S20 for string variables irrespective of the specifications in the lines or svars operands.

The section which immediately follows the last one output is then output. When the last line in the work file or the last string variable (#S20) is reached then entering + again simply causes EDT to output the last line once more and request a scrolling instruction.

+n — Switches to V mode and the range is set to 0.0001-9999.9999 for lines and #S01-#S20 for string variables irrespective of the specifications in the lines or svars operands.
EDT then outputs the section which starts n lines after the last output line in the work file (the same applies equivalently to string variables). When the last line in the work file or the last string variable (#S20) is reached then entering +n again simply causes EDT to output the last line once more and request a scrolling instruction.

\- — Irrespective of the specifications in the lines or svars operands, the range 0.0001-9999.9999 is set for lines and #S01-#S20 is set for string variables (Caution: unlike in the case of +, there is no switch to V mode).

The section which immediately precedes the last one output is then output. When the first line in the work file or the first string variable (#S00) is reached then entering – again simply causes EDT to output the section which starts with the first line once more and request a scrolling instruction.

–n Irrespective of the specifications in the lines or svars operands, the range 0.0001–9999.9999 is set for lines and #S01–#S20 is set for string variables (Caution: unlike in the case of +, there is no switch to V mode). EDT then outputs the section which starts n lines before the first line of the last section to be output in the work file (the same applies equivalently to string variables). When the first line in the work file or the first string variable (#S00) is reached then entering –n again simply causes EDT to output the section which starts with the first line once more and request a scrolling instruction.

0 Terminates the output of the current range. If more than one range was specified for output in the @PRINT statement then EDT moves on to the next range.

If in the case of section-by-section output, a long line no longer fits on the screen then the section is terminated before this line (if it already contains lines) even if the screen has not been filled. If the very first line is longer than the screen permits then this line is output as a separate section. Output of the line in interactive mode is only interrupted if the operating system's overflow monitoring is active.

All output is converted from the character set used by the work file or string variable into the character set defined for SYSOUT or SYSLST.

If characters are found which do not correspond to a valid character in the target character set then these are replaced by a substitute character if such a character has been specified (see @PAR SUBSTITUTION-CHARACTER). Otherwise blanks are used.

A line feed is inserted in the output after every 132 characters (or 160 characters if job switch 6 is set) when outputting to SYSLST. Output to SYSOUT is wrapped as specified in the WROUT or WRTRD macros. However, line feeds always occur at a character boundary.

Note

Unlike in compatibility mode, the entry of a command in response to the scroll prompt is always rejected and the request to enter a scrolling instruction is issued again. To enter a command, it is therefore first necessary to terminate @PRINT by entering 0. In interactive mode, the output from the @PRINT statement is generated by WROUT or WRTRD and, in the case of @PRINT...E, both together. There may therefore be little point in redirecting SYSOUT to a file, in particular in view of the problems relating to character sets as described in the section “System files” on page 149.

9.92 @PROC (format 1) – Switch work files

This format of the @PROC statement causes EDT to switch to another work file. This work file then becomes the current work file.

Operation	Operands	L mode
@PROC	procnr [comment]	

procnr The number of the work file (1..22) that EDT is to switch to.

comment Any text as a comment.

The work file that is switched to using this statement this work file continues to be the current work file until an @END statement is issued in order to return to the previous current work file or a further @PROC or @SETF(procnr) statement causes a switch to another work file.

When @PROC is used to switch to a work file, EDT remembers the work file it has just left together with its predecessors (nested work files), i.e. the @END statement returns to the exited work file(s). In contrast, if the user changes the work file with @SETF(procnr) then EDT deactivates the nesting of work files before switching to the new work file and any subsequent @END statement always returns to work file 0 (unless the user has already switched to work file 0 with @SETF). The nesting depth is 22. After this, error message EDT4962 is issued.

If the number of the current work file is specified in the @PROC statement then the statement is rejected with the message EDT4909. An active work file cannot be specified as the current work file. If the user attempts to do so, the message EDT4959 is issued.

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 ------------------------------------------------------- (1)
5.     @PROC 10 --------------------------------------------------- (2)
1.     @ @SET #S10 = CHAR #I6
2.     @ @ON #S10:2-2: DELETE '0'
3.     @ @IF .TRUE. : @GOTO 2
4.     @END
5.     @PROC 12 --------------------------------------------------- (3)
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.
```

(1) Processing returns to work file 0. Work file 6 contains 6 EDT statements including a @DO 10 and a @DO 12. However, these two work files do not yet exist. Consequently, entering a @DO 6 at this point would result in an error.

(2) Work file 10 is set up. It contains an EDT procedure which transfers the value stored in `#I6` to `#S10` in printable form and deletes any leading zeros.

(3) Work file 12 is set up. It contains an EDT procedure which transfers the content of the line number variable `#I6` to `#S12` in printable form.

Example 2

```
1.     @SET #I4 = 1
1.     @PROC #I4 ------------------------------------------------------ (1)
1.     @4.00
4.00   @ @SET #I4 = #I4 + 1 ------------------------------------------- (2)
4.01   @ @IF #I4 > 4 : @GOTO 8
4.02   @ @PROC #I4
4.03   @ @PROC ------------------------------------------------------- (3)
4.04   @ @GOTO 4
4.05   @8.00
8.00   @ @SET #I4 = #I4 - 1
8.01   @ @END -------------------------------------------------------- (4)
8.02   @ @IF #I4 = 2 : @RETURN
8.03   @ @PROC ------------------------------------------------------- (5)
8.04   @ @GOTO 8
8.05   @END
1.     @DO #I4 -------------------------------------------------------- (6)
<02>
<03>
<04>
<03>
<02>
1.
```

(1) It is also possible to switch to a work file by means of an integer variable. The integer variable must be between 1 and 22.

(2) Executing work file 1 causes processing to switch to work files 1 and 4. The relevant work file number is always passed in #I4.

(3) @PROC is issued to check which work file is currently being used. Thus if work file #I4 is being executed at this point, the associated #I4 is recorded.

(4) A single @END is used to ensure that processing returns to the work file located one level higher in all cases.

(5) After the return to this work file, the work file that is currently being used is checked again. As a result, a descending sequence of work file numbers from 3 to 2 is output.

(6) An integer variable can also be used to call a work file for the first time.

9.93 @PROC (format 2) – Output information about work files

This format of the @PROC statement is used to output the number of the current work file, the numbers of all the free work files and the numbers of all the work files that are in use. A work file (other than work file 0) is considered to be in use if it was once or still is the current work file and has not been released with @DROP or @DELETE (format 2) in the meantime. A work file that is in use does not necessarily have to contain records.

Operation	Operands	L mode
@PROC	[**F**REE / **U**SED]	

FREE The numbers (1-22) of the work files which are not yet in use are to be output.

If no work file apart from work file `0` is in use then EDT outputs the message `EDT0907`. If all the work files are in use then there is no output.

USED The numbers (1-22) of the work files which are already in use are to be output. The lowest and highest line numbers are output for each number.

If no work file apart from work file `0` is in use then EDT outputs the message `EDT0907`.

If no operand is specified then the number of the current work file is output on the screen.

Note

In compatibility mode, a work file is not considered to be in use unless it was, at some point, the current work file, was quitted with an @END statement, a @PROC statement, a @SETF statement or, in F mode, by means of one of the statements `0..22` and has not since been released using @DROP. Consequently, in this case the current work file set with @PROC is not considered to be in use even if it contains data.

Example 1

```
   1.     @PROC ------------------------------------------------------- (1)
   <00>
   1.     @PROC 15 ---------------------------------------------------- (2)
   1.     @PROC ------------------------------------------------------- (3)
   <15>
   1.     @END -------------------------------------------------------- (4)
   1.     @PROC ------------------------------------------------------- (5)
   <00>
```

(1) The query to check the current work file returns the number <00>.

(2) Processing switches to work file 15.

(3) When the number of the work file is queried, <15 > is now output.

(4) Processing returns to work file 0.

(5) The user is now in work file 0 again.

Example 2

```
   1.     @DROP ALL --------------------------------------------------- (1)
   1.     @PROC USED -------------------------------------------------- (2)
% EDT0907 NO WORK FILES USED
   1.     @PROC 13
   1.     A
   2.     @END
   1.     @PROC USED -------------------------------------------------- (3)
   <13>     1.0000 TO    1.0000
```

(1) All the work files are released.

(2) Queries which work files are in use.

(3) After the creation of work file 13, this again queries which work files are in use.

Example 3

```
   1.     @DROP ALL ------------------------------------------------------ (1)
   1.     @PROC FREE ----------------------------------------------------- (2)
% EDT0907 NO WORK FILES USED
   1.     @PROC 13
   1.     A
   2.     @END
   1.     @PROC FREE ----------------------------------------------------- (3)
   01 02 03 04 05 06 07 08 09 10 11 12 14 15 16 17 18 19 20 21 22
```

(1) All the work files are released.

(2) Queries which work files are not in use.

(3) After the creation of work file 13, this again queries which work files are not in use.

9.94 @QUOTE – Redefine delimiter character for strings

The @QUOTE statement redefines the delimiter characters apostrophe (single quotes) and quotation mark (double quotes) (see section “Delimiter characters” on page 81).

Strings which are enclosed in these delimiters are literals which play a special role in statements (see section “Statement syntax” on page 157).

Operation	Operands	F mode, L mode
@QUOTE **@QE**	[spec] [,char]	

spec — Special character that is to be used as the *apostrophe*. When EDT starts, the value `'` is set.

char — Character that is to be used as the *quotation mark*. When EDT starts, the value `"` is set.

It is obligatory to specify one of the two operands. It is not possible to specify special characters which have a special meaning such as blanks, the semicolon in F mode, or the comma.

The characters used for *apostrophe* and *quotation mark* must be different as otherwise the statement is rejected with the message `EDT4903`. They must also be different from the wildcards as otherwise, the statement is rejected with the message `EDT5461`.

If `spec` is not one of the valid special characters then @QUOTE is rejected with the error message `EDT3952`.

Note

The specification does not apply to the operands in the @DO and @PARAMS statements.

9.95 @RANGE – Declare line range symbol

The @RANGE statement can be used to declare a symbol for line ranges.

Operation	Operands	F mode, L mode
@RANGE	[= spec = { lines / svars }]	

spec — Special character that is to declared as the line range symbol. If `spec` is not one of the valid special characters then @RANGE is rejected with the error message `EDT3952`.

lines — Line range that is to be assigned to the line range symbol.

svars — Range of string variables which are to be assigned to the line range symbol.

When EDT starts, the character `&` is defined as the line range symbol with the line range `0.0001-9999.9999`. If a new line range symbol is declared then the old one becomes invalid.

If no operand is specified then the line range symbol is canceled. There is then no line range symbol until a new one is declared using @RANGE.

9.96 @READ – Read a SAM file

The @READ statement fully or partially reads a SAM file from disk or tape into the current work file.

Operation	Operands	F mode, L mode
@READ	[file] [(ver)] [lines[,...]] [:cols[,...]:] [{ **R**ECORD**S** / **K**EY }] [**S**TRIP]	

file — Name of the SAM file that is to be read in. The name must correspond to the SDF data type <filename 1..54> or must consist of the special specification '/'.

If there is as yet no local @FILE entry for the work file then, if the statement is successful, the specified file name is entered as an implicit local @FILE entry. If the file, operand is not specified then the explicit local @FILE entry, if present, and otherwise the global @FILE entry is used as the file name (see also @FILE statement).
If neither an explicit local nor a global @FILE entry is defined (e.g. there is only an implicit local file entry) then the @READ statement is rejected with the error message EDT5484.

If the specified file does not exist or cannot be accessed as required then the statement is rejected with a corresponding error message.

If the file link name EDTSAM is assigned to a file then the user simply needs to specify '/' in order to read this file (see chapter “File processing” on page 131).

ver — Version number of the file that is to be read. If the specified version number does not match the file's version number, the message EDT0902 is output and the file is read in nevertheless.

lines — One or more line ranges that are to be read in from the SAM file. If symbolic line numbers are specified then their values are determined from the current work file and therefore usually have nothing to do with the record structure of the file specified in file.

If lines is specified together with RECORDS then lines refers to the file's logical line numbers (see RECORDS). If lines is specified without RECORDS then the result is the same as specifying lines with KEY.

If lines is not specified, the entire file is read in.

cols
: One or more column ranges which define the section to be read in from each record. The ranges may repeat and overlap. The column specifications refer to the characters in the file that is to be read in. In the case of files which are present in a Unicode character set, they do not usually correspond to the byte positions within a record. If column values which exceed the record length are specified then blanks are read into the work file in their place. If KEY is specified (or lines without RECORDS) then the column count starts after the key in the record.

  If no column range is specified then the lines are read in full.

KEY
: EDT interprets the first 8 characters of each read in record as a key. The records that are read into the work file are assigned this key as a line number and not as part of the line content. The KEY operand is the default value if lines has been specified.

  In this case, EDT checks whether a valid key is present in the first 8 characters of each line. To be valid, the key may consist only of the digits 0 to 9. Otherwise, the @ READ statement is aborted with the error message EDT4984. The records read up to this point are taken over into the work file. If a record has the key 0 then it is treated in the same way as a record with the key 1 (line number 0.0001) and the warning EDT2900 is issued.

  If lines is used to select lines then EDT assumes that the records in the file have ascending keys. If they do not, it is possible that not all the expected records will be read in.

RECORDS
: Specifies that a line range (see lines operand) is to be selected via the file's logical line numbering. The logical line number of the first line is 0.0001, that of the second line 0.0002 etc. The records that are read in are inserted at the position in the work file determined by the current line number (see below).

  If STRIP is not specified then the RECORDS keyword can be abbreviated to R. The RECORDS operand is the default value if lines has not been specified.

STRIP
: Causes any blanks at the end of each generated work file line to be deleted. If a line consists solely of blanks then all the blanks except one are deleted.

If KEY is specified (or lines without RECORDS) then lines shorter than 8 characters are ignored.

If lines or cols is specified then line numbers or column numbers may repeat, thus resulting in the corresponding lines or columns being read in several times.

If the read operation is performed without a KEY specification then the line numbers are assigned as a function of the current line number and current increment (see section “Line number assignment” on page 36).

The file is only opened temporarily during the read operation. If the file to be read in is empty, warning EDT2903 is output.

If the current work file is empty and has the character set *NONE then it is assigned the character set of the file that is to be read in. If this character set is *NONE then the work file is assigned the character set EDF03IRV.

If the work file already has a character set then the records that are to be read in are converted from the file's character set into the work file's character set. If the file that is to be read contains characters which cannot be displayed in the work file's character set then these are replaced by a substitute character if such a character has been specified (see @PAR SUBSTITUTION-CHARACTER), otherwise file read is aborted and the error message EDT5453 is output. This also applies if there are invalid characters outside of the column range that is to be read. In contrast, invalid characters outside of the line range that is to be read are ignored.

If the file is present in a Unicode character set and contains an illegal byte sequence, e.g. surrogate characters, then it will be impossible to read it even if SUBSTITUTION-CHARACTERS is specified. In this case, the read operation is rejected with the message EDT5454.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Note

If an attempt is made to use @READ to read an ISAM file then EDT issues the error message EDT1901 and sets the switch for EDT errors. It nevertheless reads the specified file by performing an internal @GET for this file.
In this case, the operands STRIP, RECORDS and KEY are ignored.

If file is specified then it is still possible, for reasons of compatibility, to abbreviate @READ to R in F mode.

9.97 @RENUMBER – Renumber lines

The @RENUMBER statement is used to renumber the lines present in the work file. The user can specify both the line number which is to accommodate the first line in the work file and the increment which is to be used for renumbering. This increment also becomes the new current increment (independently of whether it is specified explicitly or defined implicitly by means of the specified line number) The new, current line number is then calculated as the highest line number after renumbering plus the current increment value.

Operation	Operands	F mode, L mode
@RENUMBER	[line [(inc)]]	

line — This operand specifies the line number which is to contain the first line in the work file after renumbering.

If the `line` operand is not specified then the first line in the work file is assigned line number `1`.

inc — This operand specifies the new current increment. If `inc` is not specified then the increment implicitly specified by `line` is used (see section "Implicit increment assignment" on page 35).

The @RENUMBER statement cannot be used for a file opened for real processing with @OPEN (format 2).

If the work file is empty then @RENUMBER is ignored and, in particular, no new increment is set.

If the statement is issued in interactive mode and lines would be lost because the greatest possible line number is reached then the following message is output:

```
%  EDT0910 '@RENUMBER': LINES WILL BE LOST
%  EDT0911 CONTINUE PROCESSING? REPLY (Y=YES; N=NO)?
```

If the user responds N to message `EDT0911` then the @RENUMBER statement is not executed. In contrast, if the user responds `Y` to message `EDT0911` then the @RENUMBER statement is executed, excess lines are deleted and the message `EDT2904` is output.

In F mode, the file section displayed in the data window is usually retained following renumbering with the @RENUMBER statement. Only the number display changes. The only exception here is if renumbering causes lines that were previously visible in the data window to be lost at the end of the work file. After renumbering, the lost lines are no longer displayed in the data window. If renumbering causes the loss of all the lines visible in the data window then only the last line of the work file that is still present after renumbering is still displayed in the data window.

The execution of the @RENUMBER statement is not aborted if the EDT session is continued with /INFORM-PROGRAM after being interrupted with K2.

Note

The line number/increment pairs stored in an internal memory area by the @SET statement (format 6) (see @SET statement, format 6) are not modified by the @RENUMBER statement and the same also applies to any line numbers present in the copy buffer (see statement codes, C, R, M).
Once renumbering has been performed successfully using the @RENUMBER statement, the lines that were originally identified by the line numbers stored in these two areas may therefore have new line numbers.
It is not therefore usually of any value to access the line numbers stored in these two areas (using the @SET statement without operands or the statement codes A, B or O) after renumbering using the @RENUMBER statement.

9.98 @RESET – Reset EDT and DMS error switches

The @RESET statement can be used to reset EDT and DMS error switches.

Operation	Operands	F mode, L mode
@RESET		

It is only possible to reset both error switches together. It is not possible to choose between them. The statement is primarily used in EDT procedures in combination with @IF (format 1).

9.99 @RETURN – Return from EDT procedures

The @RETURN statement is used in EDT procedures to terminate the execution of the procedure and return to the point at which it was called. If the @RETURN statement is issued outside of an EDT procedure then the EDT session or, after @DIALOG, the screen dialog is terminated.

Operation	Operands	F mode, L mode
@RETURN	[message]	

message
: The `message` operand can contain any text which is passed to the calling program when EDT is called as a subroutine.

  The `message` operand may only be specified when EDT is called via the subroutine interface or on the return from an EDT procedure.

In this statement, it is obligatory for at least one blank to be entered between the statement name and any specified operands.

If the @RETURN statement is issued outside of an EDT procedure then it has the same effect as @HALT (see the @HALT statement).

If @RETURN is used in @DO or @INPUT procedures then execution of the procedure is aborted and processing continues at the point where the procedure was called. In this case, the `message` operand is ignored.

Note

A @RETURN statement in a @DO procedure also aborts any external loop, i.e. the procedure is not executed as many times as is specified by means of the start value, end value and the increment of the loop counter in the @DO statement (see section “EDT procedures” on page 64 and @DO statement).

Example 1

```
1.     @PROC 6 ------------------------------------------------------ (1)
1.     @ @SET #S1 = 'I AM #S1'
2.     @ @SET #S2 = 'I AM #S2'
3.     @ @PRINT #S1
4.     @ @RETURN ---------------------------------------------------- (2)
5.     @ @PRINT #S2
6.     @END --------------------------------------------------------- (3)
1.     @DO 6 -------------------------------------------------------- (4)
  #S01 I AM #S1
```

(1) Work file 6 is opened for processing.

(2) When the procedure is executed in work file 6 for this statement then the following statements are no longer executed.

(3) Execution of work file 6 is started.

(4) The procedure is executed.

Example 2

```
1.     AAAA
2.     BBBB
3.     CCCC
4.     @PROC 7 ------------------------------------------------------ (1)
1.     @ @PRINT ! --------------------------------------------------- (2)
2.     @ @RETURN ---------------------------------------------------- (3)
3.     @END
4.     @DO 7,!=%,$ -------------------------------------------------- (4)
1.0000 AAAA
4.
```

(1) The procedure is created in work file 7.

(2) When the statements present in the procedure are executed, the line addressed by the loop counter ! is to be output.

(3) The execution of the statements in the procedure is terminated here irrespective of whether the loop counter has reached the upper limit or not.

(4) The procedure is called. In this case, the loop counter `!` is to take on the values `1` to `3` (`%=1, $=3`). However, due to the @RETURN present in the procedure, the loop counter is not incremented.

9.100 @RUN – Call user routine

The @RUN statement is used to execute a routine written by the user (user routine) (see Subroutine Interfaces User Guide [1]).

Operation	Operands	F mode, L mode
@RUN	**E**NTRY=entry [,**M**ODLIB=modlib] [,**U**NLOAD] [,string]	

entry — Entry point for the user routine.

modlib — Name of the library containing the module which contains the entry point. The library must exist. Otherwise, the statement is rejected with the error message EDT5372.

If no module containing the entry point is found in the specified library, the system first searches in the alternative libraries BLSLIBnn then in the private task library and system task library $TASKLIB.

If no library is specified, the system first searches in the private task library and then in the system task library $TASKLIB.

If the search fails, the error message EDT5372 is issued.

UNLOAD — Specifies that the load unit which contains the entry point is to be unloaded following return to EDT. The UNLOAD operand has the same effect as a separate @UNLOAD UNIT=entry statement, i.e. it only causes an unload if the specified entry point is also the name of a load unit (see below). If, for example, the entry point has been found in an already loaded load unit with a different name then it is not possible to execute UNLOAD. In this case, the message EDT1907 is output.

string — String that is passed to the called program.

The implementation of external user routines and the way parameters are passed to them are explained in more detail in “User Routines - @RUN” [1].

If no initialization routine is defined for the user routine then the statement is rejected with the message EDT5469. If the associated initialization routine sends a return code then the statement is rejected with the message EDT5470 if the initialization routine does not support the version and the message EDT5471 if the initialization routine reports a different error.

If the execution of the @RUN statement results in a separate load operation then a UNIT name which is the same as the specified entry point is notified to the dynamic binder loader .

This UNIT name can be specified in the @UNLOAD statement in order to unload all the load units that were loaded together with the entry point. The @RUN statement's UNLOAD operand also refers to this UNIT name. If the entry point is found inside another load unit, @RUN does not therefore result in a separate load operation and the entry point can only be unloaded together with this load unit.

The @RUN statement is one of the EDT statements with security implications (see also section “Access protection” on page 99). Under certain privileged IDs, the @RUN statement is rejected. This also applies to uninterruptible system procedures in interactive mode (read from SYSDTA with RDATA, execute EDT start procedure) unless the @RUN statement is issued by the protected procedure itself (SYSDTA=SYSCMD).

Note

The entry operand, which accepts names of up to 32 characters in length, is case-sensitive.

Caution

The format of the statement and interface used to call the routine are different in Unicode and compatibility mode.

9.101 @SAVE – Write as ISAM file

The @SAVE statement fully or partially writes the content of the current work file to disk as an ISAM file.

Operation	Operands	F mode, L mode
@SAVE	[file] [(ver)] [lines[,...]] [:cols[,...]:] [{ **U**PDATE [**R**ENUMBER [line [(inc)]]] [**O**VERWRITE] }]	

file — Name of the ISAM file that is to be written. The name must correspond to the SDF data type <filename 1..54> or must consist of the special specification '/'.

If the file operand is not specified then the explicit local @FILE entry is used as the file name if present. If not, the global @FILE entry is used and, failing this, the implicit local @FILE entry (e.g. from the @GET statement) (see also @FILE statement). If there is neither a local nor a global @FILE entry then the @SAVE statement is rejected with the error message EDT5484.

If the specified file cannot be accessed as required then the statement is rejected with a corresponding error message.

If the file link name EDTISAM is assigned to a file then the user simply needs to specify '/' in order to write this file (see chapter "File processing" on page 131).

ver — Version number of the file that is to be overwritten. If an incorrect version number is specified for an existing file then the statement is rejected with EDT4985. If the file does not yet exist then this specification is ignored and version 001 of the file is written.

lines — One or more line ranges that are to be written to the ISAM file. If lines are specified more than once then they are also written more than once.

If lines is not specified, then the entire file is written.

cols — One or more column ranges which define the section to be written for each record. The ranges may repeat and overlap. The column specifications refer to the *characters* in the current work file. If column values which exceed the work file record length are specified then blanks are written to the file in their place.

If no column range is specified then the lines are written in full.

UPDATE — Specifying `UPDATE` causes the lines that are to be saved to be inserted in the ISAM file. In the ISAM file, EDT only overwrites the lines whose numbers also exist in the current work file and which are located in the range specified with `lines`. The remaining lines in the ISAM file are retained.

This operand is ignored if no ISAM file with the specified name exists.

RENUMBER — New ISAM record keys are formed for the lines that are to be saved. The line numbering in the work file remains unchanged. If lines are output more than once (due to overlaps in the range specifications) then they are also entered in the file multiple times (with different record keys).

If `RENUMBER` is not specified then the ISAM keys result from the line numbers of the lines that are to be saved. If the file is to be read subsequently with @GET ... NORESEQ and is to have the same line numbering as at the time it was saved then `RENUMBER` *must not* be specified.

If renumbering results in the maximum permitted line number (corresponding to the key length) being overwritten then the statement is aborted with error message `EDT5252`.

line — Starting number for the new ISAM record keys that are to be formed. If `line` is not specified then the value `1` is used.

inc — Increment for the new ISAM record keys that are to be formed. If `inc` is not specified then the increment implicitly specified by `line` is used (see section “Implicit increment assignment” on page 35).

OVERWRITE — An existing file of the same name is overwritten without any request for confirmation. If the specified file does not yet exist, `OVERWRITE` has no effect.

If neither `UPDATE` nor `OVERWRITE` is specified and if a file with the same name already exists then, in interactive mode, EDT issues the query:

```
%  EDT0903 FILE 'file' IS IN THE CATALOG, FCBTYPE = fcbtyp
%  EDT0296 OVERWRITE FILE? REPLY (Y=YES; N=NO)?
```

If the user answers the message with Y then the existing file is overwritten as an ISAM file with the content of the current work file. In contrast, if the user answers N then the file is not written and the message EDT0293 is output.
In batch mode, the file is always overwritten.

If an existing file is overwritten by @SAVE without the UPDATE operand then the file type and file attributes may change. The file is written as an ISAM file with default attributes (e.g. variable record length unless a corresponding /SET-FILE-LINK command with the file link name EDTISAM and the divergent attributes has previously been issued (see chapter "File processing" on page 131). Files of the type PAM or BTAM cannot be overwritten.

The file is only opened temporarily during the write operation.

The character set used for the write operation depends on whether the file is overwritten, created or extended (see the UPDATE operand.

If the file is overwritten or created then the data is written in the work file's character set and this character set is entered for the file in the catalog.

If the file is extended then the data is converted from the work file's character set into the character set specified in the file's catalog entry.
If the value *NONE is entered for the file in the catalog then EDF03IRV is used (see also section "Character sets" on page 47). If the work file contains characters which are invalid in the character set of the file that is to be written then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the file is not written and the error message EDT5453 is output.
This does not apply to invalid characters outside of the line or column range that is to be written. These are ignored.

If the work file contains lines that are too long for the file that is to be written (e.g. if the file has a fixed record length) or if the conversion operation creates any such records (possible in the case of Unicode character sets), then the write operation is aborted with the message EDT5444.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Note

If file is specified then it is still possible, for reasons of compatibility, to abbreviate @SAVE to S in F mode.

9.102 @SCALE – Output column counter

The @SCALE statement activates or deactivates the display of a column counter (horizontal ruler) for the current work file in the work window (see also section "The work window" on page 103).

Operation	Operands	F mode
@SCALE	[**ON** / OFF]	

ON — Activates the display of the column counter (default value).

The column counter is displayed as the first line after an information line, if one is present and displays the current column numbers for the work file (e.g. after the work window has been moved horizontally).

If a tabulator has been defined (see @TABS statement), an additional screen line is displayed in which the current position of the tabulator is displayed with `'I'`. In this line, the tabulator character itself is depicted in the statement code column.

OFF — Deactivates the display of the column counter and any displayed tabulator positions.

When an EDT session starts, the column counter display is deactivated for all the work files.

The activation and deactivation of the column counter display applies at *work file* level. If the work file is simultaneously displayed in multiple data windows on the screen then the statement therefore applies in both data windows.

If the data window is too small to display at least one data line in addition to the column counter and any displayed information line or tabulator line then some of the displays are hidden. The data window must then be enlarged appropriately. The hidden displays are then shown again in the sequence: information line, column counter, tabulator display.

In `EDIT-LONG` mode (see the @EDIT statement), the column counter is not output. In HEX mode, the @SCALE statement has no effect. However, in both cases, the setting becomes effective if the corresponding mode is exited.

The @PAR SCALE statement can be used instead of @SCALE and has the same functionality. Furthermore, @PAR SCALE can be used for a specific work file or globally for all the work files and is also permitted in L mode and therefore in EDT procedures.

Example

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
   2.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   3.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································································
:                                                                              :
:                                                                              :
:                                                                              :
@scale on·····················································0001.00:00001(00)
```

A column counter is requested in order to check the column numbers.

```
        ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   1.00 BERGER   ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····················
   2.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   3.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································································
```

9.103 @SDFTEST – Syntax check by SDF

The @SDFTEST statement is used to check whether a line range contains syntactically correct SDF commands or syntactically correct SDF statements.

A program name can be set for the check of the SDF syntax of SDF statements.

If the SDF option `GUIDANCE=MIN|MED|MAX` is set then the user is taken to the SDF correction dialog if incorrect SDF syntax is detected.

If the user aborts the correction dialog or if the dialog is not possible then the error message `EDT4310` is output if the SDF syntax is incorrect.

If the SDF syntax is correct or has been corrected then the text is taken over into the work file. The format in which the statement is taken over is determined by the SDF `LOGGING` option (see the description of the `/MODIFY-SDF-OPTIONS` command and the description in the SDF User Guide [6]).

The current SDF settings apply. These can be modified with `/MODIFY-SDF-OPTIONS`.

Operation	Operands	F mode, L mode
@SDFTEST	[lines[,...]] [**PROG**RAM [= progname [{ **I**NTERNAL | **E**XTERNAL }]]]	

lines — One or more line ranges in which the SDF syntax of SDF commands and, if required, also of SDF statements is to be checked.

If `lines` is not specified then the SDF syntax of all the SDF commands and, if required, SDF statements in the work file is checked.

PROGRAM= — Causes the SDF syntax of SDF statements to be analyzed.

If `PROGRAM` is not specified then only the SDF syntax of SDF commands is analyzed.

progname — Name of the program whose statements are to be subjected to a syntax check in accordance with the SDF syntax file hierarchy.

If `progname` is not specified then the predefined name set by the @PAR SDF-PROGRAM statement is used. If no name has been preset then the @SDFTEST statement is rejected with the message `EDT5320`. If the program name is not known in the current SDF syntax file hierarchy then the @SDFTEST statement is rejected with the message `EDT5321`.

INTERNAL The program name is the maximum 8-character internal name. The internal name can be ascertained using SDF-A if it does not correspond to the name of the program.

EXTERNAL The program name is the maximum 30-character external name (e.g. LMS, SDF-A or HSMS).

If neither INTERNAL nor EXTERNAL is specified as the name type then the name type set as the default value in the @PAR SDF-NAME-TYPE statement is used. When EDT starts, the name type INTERNAL is set as the default.

EDT differentiates between 3 types of record content:

1. Records which start with one (and only one) '/' in column 1:

   These are checked for command syntax in accordance with the SDF syntax file hierarchy. Admissibility in terms of privileges and system environment is determined by the current user and the current environment.

2. Records which start with '//':

   These are passed to SDF for a statement check if PROGRAM has been specified.

3. Other data lines:

   Records which do not start with either '/' or '//' are ignored.

Lines which start with '/' and have a continuation character ('-') as their final character are chained with the next line provided that this also starts with '/'. The two lines are then passed to SDF together when the @SDFTEST statement is executed. The continuation lines do not have to be present in any of the specified line ranges. It is sufficient for the first line to be present in one of the specified line ranges. If PROGRAM is specified then this procedure also applies to lines that start with '//'.

The checked command or statement overwrites the old command or statement in the work file together with all the continuation lines. If the command or statement is modified during the SDF check (for example because LOGGING=INVARIANT was set in a preceding /MODIFY-SDF-OPTIONS command) then the affected lines are reformated and split into multiple continuation lines if necessary. The continuation character is set in the 72nd column. If necessary, the following lines are renumbered. Line numbers are assigned using the procedure Insertion between two lines (see section “Line number assignment” on page 36). If it is not possible to insert the lines generated by SDF then the statement is aborted with the message EDT5364 or EDT5365.

In F mode, the message EDT0285 is issued after all the checks have been performed if no errors have occurred or if processing was continued after an error. In L mode, the SDF output for statements (but not for commands) is sent to SYSOUT.

If a checked command or statement contains errors and the correction dialog is unsuccessful, EDT outputs the message EDT4310. In interactive mode, the user is also asked whether the check is to be continued.

```
%  EDT4310 SDF: SYNTAX ERROR IN LINE (&00)
%  EDT0911 CONTINUE PROCESSING? REPLY (Y=YES; N=NO)?
```

If the user responds N to the message in interactive mode then the @SDFTEST statement is interrupted with the message EDT5324 and, in F mode, the incorrect line is displayed at the topmost position in the window. In contrast, if the user responds Y then the syntax check continues with the next line which has not yet been examined. In batch mode, the syntax check is always continued.

If the work file containing the lines to be checked has a character set other than EDF03IRV then it is necessary to take note of certain special characteristics of SDF use. In particular, characters which do not belong to the EBCDIC kernel are naturally only permitted in literals or comments. Furthermore, SDF always conducts the correction dialog in the character set defined using /MODIFY-TERMINAL-OPTIONS and also always interprets the byte sequences passed to it in this character set.

Consequently, before passing the statements or commands to SDF, EDT converts them into the character set defined with /MODIFY-TERMINAL-OPTIONS if the currently set GUIDANCE-MODE does not make it possible to conduct a correction dialog. If this operation is unsuccessful, then the @SDFTEST statement is rejected with the message EDT5327.

If no correction dialog is possible then EDT uses other (less constraining) rules for the conversion. If the work file has an EBCDIC character set then this is used without conversion. If the work file possesses an ISO character set then the corresponding EBCDIC reference character set is used. In all other cases, EDT uses the character set UTFE. If conversion is not possible, then the @SDFTEST statement is aborted with the message EDT5327.

If SDF returns data then this is converted into the work file's character set. If this is not possible, then the @SDFTEST statement is aborted with the message EDT5453.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Note

If GUIDANCE=EXPERT is set then EDT displays any errors reported by SDF only in the form EDT4310. To permit a more precise error analysis, it is advisable to set GUIDANCE=MIN, MED or MAX.

Passwords and other operands which have been defined using OUTPUT=SECRET-PROMPT are replaced by P if the GUIDANCE setting is MIN, MED or MAX.

SDF does not recognize incorrect operands in ISP commands.

If the character set UTFE has been set in /MODIFY-TERMINAL-OPTIONS then the screen layout for the SDF correction dialog is shifted if characters not present in EDF03IRV occur. This is due to the fact that SDF does not currently support Unicode. However, it does not usually cause any functional restrictions.

Commands and statements may not exceed a maximum length of 16379 bytes either on input or on output. Otherwise, the @SDFTEST statement is rejected with the message EDT5325 or EDT5326.

9.104 @SEARCH-OPTION – Set default value for searching with @ON

On the one hand, the @SEARCH-OPTION statement can be used to specify whether the @ON statement is to differentiate between uppercase and lowercase in the search term when searching for strings and, on the other, it can be used to define a global column range to which the search is restricted if the @ON statement contains no explicit column range specification.

Operation	Operands	F mode, L mode
@SEARCH-OPTION	{ **CASELESS**-SEARCH [= { **ON** | **OF**F }] | **COL**UMN-RANGE [= cols] } [,...]	

CASELESS-SEARCH=

The operand specifies whether a distinction is to be made between uppercase and lowercase characters in the search term when searching with @ON.

ON The search with @ON does not consider whether the characters in the located text match those in the search term in terms of uppercase and lowercase notation, i.e. if `'string'` is defined as the search term then the strings `'String'`, `'STRING'` or `'StrIng'` are all identified as hits (see also section “Searching with @ON” on page 78).

OFF The uppercase/lowercase notation of a character is taken into consideration during the search. This is the default setting when EDT is started.

COLUMN-RANGE=

The operand provides a global specification for the column range to which the search is to be restricted if the @ON statement contains no explicit column range specification.

cols Contiguous column range to which searches using the @ON statement are to be limited. If the range specification contains only a single column specification, this indicates the range from the specified column through to 32768. If the first operand is greater than the second then the statement is rejected with the message `EDT3922`.

If no column range is specified then the setting is reset to the value defined at EDT start time, i.e. `1–32768`.

EDT uses the system component XHCS when assigning lowercase characters to uppercase. Which characters are treated as an uppercase/lowercase pair therefore depends on the definition of the associated character set attributes in XHCS.

The option `CASELESS-SEARCH=ON` is effective independently of the setting made with @PAR LOWER. Consequently, in F mode with @PAR LOWER=OFF hits may be displayed in which the located characters are depicted as smudge characters.

9.105 @SEPARATE – Perform line break

The @SEPARATE statement breaks the specified lines into multiple lines. The point at which the break takes place is specified by a separator character or by a column position.

Operation	Operands	F mode, L mode
@SEPARATE	[lines[,...]] [**AT** { strchar / col }]	

lines — One or more line ranges in which line breaks are to be inserted. If the `lines` operand is not specified then all the lines in the file are processed.

AT — This operand introduces the definition of the point where the lines are to be broken.

If the `AT` operand is not specified then the record separator defined as the default using @PAR SEPARATOR determines the break position (see the @PAR SEPARATOR statement). If no default record separator has been defined then the error message `EDT4952` is output.

strchar — This operand specifies the record separator character for line breaks. It may be any character which must be specified in single quotes. The character can also be specified in the form of a substitute representation for Unicode characters.

col — This operand specifies the number of the column at which the line break is to be performed.

If the break is performed using a record separator character then no break is inserted in lines in the specified line range which do not contain this record separator.

In contrast, if a line in the specified line range contains one or more record separator characters then the line is searched through from left to right for the first occurrence of the record separator character.

All the characters before the first record separator character remain in the original line while all the characters after the record separator character (including any other record separators that may be present) are inserted as a new line in the work file.
The record separator character at which the line break is performed is removed, i.e. it is no longer present either in the original or in the newly inserted line.
If the newly inserted line itself contains further record separator characters then the procedure described above is repeated for this line. This operation continues until a newly inserted line contains no further record separator characters.

If the line contains several record separator characters which immediately follow one another or if the line starts or ends with one or more record separator characters then empty records (record length=0) are inserted.

If the point of the break is defined by means of a column number then all the characters as of this position (including the character in the specified column) are separated from the original line and inserted as a new line in the work file. If the new line still contains the same number as or more columns than are specified in the col operand then this line is itself broken at column `col`. This operation is repeated until the newly inserted line has fewer columns than are specified in the `col` operand.

If the original line possesses fewer columns than are specified in the col operand or if `col=1` is specified in the statement then the line is not modified.

The lines created as a result of the line break operation are numbered using the procedure Insertion between two lines (see section "Line number assignment" on page 36).

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

Example 1

A printed list is to be made narrower:

```
@SEPARATE 5-100 AT 41
```

Lines 5 to 100 are shortened to a length of 40 characters. The remaining characters of each line are inserted after each line in the file.

Example 2

Records contain line feed characters (`= U'000A'` in `UTF16`) which are to be evaluated (for the purposes of this example, it is assumed that the character `%` has been declared as the escape character for the substitute representation of Unicode characters, by means of the @PAR ESCAPE-CHARACTER statement):

```
@SEPARATE & AT '%U000A'
```

9.106 @SEQUENCE (format 1) – Perform line numbering

The @SEQUENCE statement (format 1) causes EDT to write a number in each line of a contiguous line range.
A predefined number consisting of a maximum of 8 digits (possibly with leading zeros) is written to the first line of the line range. This also defines the number of digits in all the following numbers. All the following numbers are given by the total of the preceding number plus the predefined increment. If this process would result in a number containing more digits than are present in the starting number then only the same number of digits from the right as are present in the starting number are used.

This statement overwrites any content in columns in which the numbers are written.

Operation	Operands	F mode, L mode
@SEQUENCE	[{lines | svars}] [: [col] [: [n1] [(n2)]]]	

lines — EDT writes a number to each line in the specified line range.

svars — EDT writes a number to each string variable in the specified range of string variables.

col — The operand specifies the column which is to accommodate the first digit of the number that is to be written. If a line in the specified line range has fewer columns than are specified in the `col` then the columns between the previous line end and the column `col` are filled with blanks.

If the `col` operand is missing, EDT writes the first digit in column 73.

n1 — This operand specifies the integer value that EDT is to write as a decimal number in the first line of the relevant line range. The `n1` operand may consist of a maximum of 8 digits (possibly with leading zeros). The numbers that are written in the following lines have the same number of digits.

If the `n1` operand is missing, EDT writes the number `00000100` in the first relevant line.

n2 — This operand specifies the increment (as an integer value) for the formation of the following numbers. Each of these numbers consists of the sum of the preceding number and the increment. During this process, only as many digits are used starting from the right as are present in the starting number `n1`.

If the `n2` operand is missing, EDT uses the value `100` as the increment.

If neither the `lines` nor the `svars` operand is specified then EDT writes a number in every line of the current work file.

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

Note

The choice of starting value and increment determines whether the sequence of numbers generated by the @SEQUENCE statement (format 1) is ascending, descending or constant. For example, the starting value `0100` and increment `100` result in the ascending sequence of numbers `0100`, `0200`, `0300`, etc., at least up to the value `9900`.
After that, the value reverts to `0000` and numbering continues with `0100`, `0200` etc.
If the starting value `999` is selected in combination with the increment `998`, then the descending sequence `999`, `997`, `995` etc. is obtained. When the value `001` is reached, the next number is `999` again and the sequence repeats from the beginning.

In the example above, the same result could be obtained by setting the increment to `3998`, for example, because the leading `3` is omitted in each newly formed number.
An alternating sequence of numbers can be obtained, for example, by setting the starting value `3` and the increment `5: 3`, `8`, `3`, `8`, etc. The easiest way to obtain a constant sequence is to set the increment `0`.

9.107 @SEQUENCE (format 2) – Adopt line numbers

The @SEQUENCE statement (format 2) causes EDT to write the associated line number in each line of a contiguous line range. The line number is written as an 8-digit number without a decimal point. If necessary, the number is filled with zeros at the right and left. EDT overwrites any content in the 8 columns in which it writes the line numbers.

Operation	Operands	F mode, L mode
@SEQUENCE	[lines] : [col] : **L**INE	

lines — EDT writes the associated line number in each line of the specified line range. If the `lines` operand is missing, EDT writes the associated line number in each line of the current work file.

col — The operand specifies the column which is to accommodate the first digit of the associated line number. If a line in the specified line range has fewer columns than are specified in the `col` operand then the columns between the previous line end and the column `col` are filled with blanks.

If the `col` operand is missing, EDT writes the first digit of the line number in column 73.

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

Example

```
   0.00 THE<·····························································
   1.11 SEQUENCE<·····················································
  88.76 IS<···························································
  88.76 OFTEN<······················································
5555.00 VERY<·························································
9876.54 IMPORTANT<···················································
9877.54 ·····························································
```

```
sequence :20: line·····························0000.00:00001(00)
```

The associated line number is to be written starting at column 20 in every line of the work file.

```
   0.00 THE                00000035<·····································
   1.11 SEQUENCE           00011110<·····································
  88.76 IS                 00887610<·····································
  88.76 OFTEN              00887620<·····································
5555.00 VERY               55550000<·····································
9876.54 IMPORTANT          98765432<·····································
9877.54 ·····················································································
```

The line numbers have been written as 8-digit numbers without a decimal point starting at column 20. The line numbers have been filled on the right and left with zeros as required.

9.108 @SEQUENCE (format 3) – Check line numbers

The @SEQUENCE statement (format 3) causes EDT to examine the content of a column or contiguous range of columns in each line of a contiguous line range. In the case of Unicode character sets, it interprets the string located there on the basis of its `UTF16` coding. Otherwise, it interprets the string as a binary number in accordance with the coding corresponding to the work file's character set. If the column to be examined is located to the right of the end of the line then EDT considers the column content to be a blank.
EDT checks whether the identified binary numbers form an ascending sequence. It outputs all the lines in which it identifies a binary number which is equal to or smaller than that of the preceding line. In interactive mode, the output is written to `SYSOUT` and in batch mode it is written to `SYSLST`.

Operation	Operands	F mode, L mode
@SEQUENCE	[lines | svars] : [col] : **C**HECK [int]	

lines — The specified column range is examined in each line in the specified line range.

svars — The specified column range is examined in each string variable in the specified range of string variables.

col — The operand specifies the column containing the first character that is to be checked. If all or part of the column range that is to be checked is located after the end of the line then a blank is taken as the value of every column located after the end of the line.

If the `col` operand is missing, EDT starts the check in column 73.

int — This operand specifies the number of columns that are to be considered (`1..8`). If the `int` operand is missing, EDT examines 8 columns.

If a range of string variables is specified then either a Unicode character set or a 7 or 8-bit character set must be defined for all the string variables. If this is not the case, the error message `EDT5473` is output and the execution of the statement is aborted.
If neither the `lines` nor the `svars` operand is specified then EDT checks every line of the current work file.

If the statement is interrupted with K2 and the EDT session is continued with `/INFORM-PROGRAM` then the processing of the statement is aborted and message `EDT5501` is output.

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·····································0001.00:00001(00)
```

EDT is to check all lines to determine whether the contents of columns 2 to 3 form an ascending sequence.

```
   4.0000 16 LINE 4
   6.0000 01 LINE 6
   8.0000 97 LINE 8
 %PLEASE ACKNOWLEDGE
```

EDT outputs all the lines that deviate from the ascending sequence.

```
   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·····································0001.00:00001(00)
```

Now, only the content of the 1st column in the first four lines is to be used for the check.

```
   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). In particular, it should be noted that the sequence is not considered to be ascending if identical values are identified. This occurs here in line 3.

9.109 @SET (format 1) – Supply values for integer variables

This format of the @SET statement is used to assign values to integer variables. This value may be the result of an expression, may be obtained by converting a printable number or the content of a line number variable into an integer value, or may take the form of a line length or the binary value of a string.

Operation	Operands	F mode, L mode
@SET	ivar = { intex lvar **L**ENGTH line **L**ENGTH svarex **S**UBSTR string **ST**RING string [,**C**ODE = name] }	

ivar — Integer variable (#I0..#I20) which is to be supplied with a value.

intex — Integer expression. If the maximum negative or positive value (-2^{31}, $2^{31}-1$) is exceeded on an arithmetic operation then the statement is rejected with the message EDT4946.

lvar — Line number variable (#L0..#L20) whose value is to be assigned to the integer variable. When converted into an integer value, the content of the line number variable is multiplied by 10000 and then assigned.

LENGTH line — Line number of a line whose length is to be assigned as a value to the integer variable. If the line is empty then the value 0 is assigned. If the line does not exist then, for reasons of compatibility, the value 0 is again assigned to the integer variable.

LENGTH svarex — String variable whose length is to be assigned as a value to the integer variable.

SUBSTR string

String specifying an integer value to be assigned to the integer variable. Any plus or minus sign present in the string is taken into account during conversion. If the sign is missing then + is assumed. If the string contains blanks then these are eliminated during conversion.

If the string is not an integer value then the statement is rejected with the error message EDT5477. If the integer value is not in the permitted range (-2^{31}, $2^{31}-1$) then the statement is rejected with the message EDT4946. Empty strings are not permitted and result in the error EDT3907.

STRING string

String. The binary value of the first 4 bytes of the string is assigned to the integer variable. If the string contains fewer than 4 bytes then it is left-filled with zeros.
Empty strings are not permitted and result in the error EDT3907.

name

Character set in which the string is to be interpreted. The string is converted into this character set before being assigned. The first 4 bytes are then assigned without regard for character boundaries.
If the string contains characters which are invalid in the target character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the statement is rejected with the error message EDT5453.

If the operand is not specified then the character set of the specified string (which depends on the source) is used.

In this statement, the statement name may be omitted entirely. In F mode, it is also permissible to omit the statement symbol.

Example

```
   1.     @SET #I0 = SUBSTR '123' ------------------------------------- (1)
   1.     @SET #L0 = 1.01 --------------------------------------------- (2)
   1.     @SET #I1 = #L0 ---------------------------------------------- (3)
   1.     @CREATE 1 'AB' ---------------------------------------------- (4)
   1.     @SET #I2 = LENGTH 1 ----------------------------------------- (5)
   1.     @SET #I3 = 2124 + #I0 + #I1 - #I2 --------------------------- (6)
   1.     @SET #I4 = STRING '123' ------------------------------------- (7)
   1.     @SET #I5 = STRING 'A',CODE=UTF16 ---------------------------- (8)
   1.     @STATUS = I ------------------------------------------------- (9)
#I00= 0000000123 #I01= 0000010100 #I02= 0000000002
#I03= 0000012345 #I04= 0015856371 #I05= 0000000065
#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
```

(1) The value `123` is assigned to the integer variable `#I0`.

(2) The value `0001.0100` is assigned to the line number variable `#L0`.

(3) The value `10100`, line number `1.01 * 10000`, is assigned to the integer variable `#I1`.

(4) Line 1 is created with the content `AB`.

(5) The value `2`, the length of line 1, is assigned to the integer variable `#I2`.

(6) The value of the expression is assigned to the integer variable `#I3`.

(7) The value `X'00F1F2F3' = 15856371` is assigned to the integer variable `#I4`.

(8) The value which corresponds to the Unicode code position `'A'` is assigned to the integer variable `#I5`.

(9) The content of the integer variable is output.

9.110 @SET (format 2) – Supply values for string variables

This format of the @SET statement is used to assign values to string variables. This value can be the binary value of an integer variable, a line number or the name of a string variable. It is also possible to assign a string. In both cases, a new string variable is created.

Operation	Operands	F mode, L mode
@SET	svarex { = **I**NTERNAL { ivar / line / svar } / = string [,**C**ODE=name] }	

svarex String variable (#S0..#S20) which is to be supplied with a value.

INTERNAL ivar
Integer variable (#I0..#I20) whose content is to be supplied to the string variable in binary form. The result is not usually printable. The character set is EDF041.

INTERNAL line
Line number which is to be assigned to the string variable as a binary value. In this case, each of the 8 digits in the line number are assigned to a half byte in binary form. The result is not usually printable. The character set is EDF041.

INTERNAL svar
String variable (#S0..#S20) whose name is to be supplied to the string variable as a value. The character set is EDF041.

string String which is to be assigned to the string variable.

name Character set which is to be assigned to the string variable. The string string is converted into this character set before being assigned. If it contains characters which are invalid in the target character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the statement is rejected with the error message EDT5453.
If the operand is missing then the string variable is assigned the character set of the string that is to be supplied to it.

In this statement, the statement name may be omitted entirely. In F mode, it is also permissible to omit the statement symbol.

Example

```
1.     @SET #I0 = -2122153084 ---------------------------------------- (1)
1.     @SET #S0 = INTERNAL #I0 --------------------------------------- (2)
1.     @SET #S1 = INTERNAL 4081.4082 --------------------------------- (3)
1.     @SET #S2 = INTERNAL #S0 --------------------------------------- (4)
1.     @SET #S3 = 'ABC' ---------------------------------------------- (5)
1.     @PRINT #S00.-#S03 --------------------------------------------- (6)
  #S00 abcd
  #S01  a b
  #S02 #S00
  #S03 ABC
```

(1) The value `-2122153084` is assigned to the integer variable `#I0`.

(2) The value `X'81828384'` is assigned to the string variable `#S0`.

(3) The value `X'40814082'` is assigned to the string variable `#S1`.

(4) The value `'#S00'` is assigned to the string variable `#S2`.

(5) The value `'ABC'` is assigned to the string variable `#S3`.

(6) The string variables `#S00-#S03` are output.

9.111 @SET (format 3) – Supply values for line number variables

This format of the @SET statement is used to assign values to line number variables. This value may consist of: a line number specification, the value of an integer variable, the specification of a line number as a string or the binary value of the first 4 bytes in a string.

Operation	Operands	F mode, L mode
@SET	lvar = { line | ivar | **S**UBSTR string | **ST**RING string[,**C**ODE=name] }	

lvar Line number variable (#L0..#L20) which is to be supplied with a value.

line Line number which is to be assigned to the line number variable.

ivar Integer variable (#I0..#I20) whose content is to be supplied to the line number variable in converted form. In this case, the permitted range of values for ivar is 1 to 99999999. This results in the line numbers 0.0001 to 9999.9999. On conversion, therefore, the value of the integer is divided by 10000 and the resulting value is assigned to the line number variable. If the value of the integer variable is outside of the valid range then the statement is rejected with the message EDT5475.

SUBSTR string

String which is converted into a line number and assigned to the line number variable. If the string does not represent a valid line number then the statement is rejected with the message EDT5477. If the string contains blanks then these are eliminated during conversion.
Empty strings are not permitted and result in the error EDT3907.

STRING string

String whose binary value is interpreted as a line number and assigned to the line number variable. In this case the 8 half bytes in the first 4 bytes of the string are each interpreted as a decimal digit.

If the string contains fewer than 4 bytes after conversion then it is left-filled with zeros.

If the binary value of one of the half bytes in the first 4 bytes of the string does not correspond to a decimal digit 0..9 then the statement is rejected with the message EDT4928.
Empty strings are not permitted and result in the error EDT3907.

name
: Character set in which the string is to be interpreted. The string is converted into this character set before being assigned. The first 4 bytes are then assigned without regard for character boundaries.

  If the string contains characters which are invalid in the target character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the statement is rejected with the error message EDT5453.

  If the operand is not specified then the character set of the specified string (which depends on the source) is used.

In this statement, the statement name may be omitted entirely. In F mode, it is also permissible to omit the statement symbol.

Example

```
   1.     @SET #L0 = 1.01 ------------------------------------------- (1)
   1.     @SET #I0 = #L0 -------------------------------------------- (2)
   1.     @SET #L1 = #I0 -------------------------------------------- (3)
   1.     @SET #L2 = SUBSTR '1.01' ---------------------------------- (4)
   1.     @SET #L3 = STRING X'00010100' ----------------------------- (5)
   1.     @STATUS = L ----------------------------------------------- (6)
#L00=   1.0100    #L01=   1.0100    #L02=   1.0100
#L03=   1.0100    #L04=   0.0000    #L05=   0.0000
#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) The value 0001.0100 is assigned to the line number variable #L0.

(2) The value 10100, line number 1.01 * 10000, is assigned to the integer variable #I1.

(3) The value 0001.0100 is assigned to the line number variable #L1.

(4) The value 0001.0100 is assigned to the line number variable #L2.

(5) The value 0001.0100 is assigned to the line number variable #L3.

(6) The content of the line number variable is output.

9.112 @SET (format 4) – Store values of variables

This format of the @SET statement is used to insert the contents of an integer variable, the name of a string variable or the contents of a line number variable as of a given column in printable form in a work file line or string variable.

Operation	Operands	F mode, L mode
@SET	{ svarex | lvar } [,col] = **C**HAR { ivar | svar | lvar1 }	

svarex — String variable (#S0..#S20) in which a value is to be inserted. Any characters in the corresponding positions are overwritten.

lvar — Line number variable (#L0..#L20) specifying the line to which a value is to be written. If the line does not yet exist, it is created. Any characters in the corresponding positions are overwritten.

col — Column as of which the line or string variable is to be written. The default value of col is 1. If col is located after the end of the line then the line is filled with blanks up to the column col.

CHAR ivar — Integer variable (#I0..#I20) whose content is to be inserted as a string in the line as of column col. The conversion produces an 11 character long, printable number starting either with a blank or a minus sign depending on whether the integer is positive or negative.

CHAR svar — String variable (#S0..#S20) whose name is to be inserted in the specified line or string variable.

CHAR lvar1 — Line number variable whose value is to be inserted in printable form in the specified line or string variable.

Converting 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. In this case, leading zeros are replaced by blanks.

If the insertion operation causes the line or string variable to exceed the maximum length of 32768 then the statement is rejected with the message EDT5474.

If the information is inserted in a line then the character set depends on the work file. If the current work file already has a character set then the value is inserted in this character set. If the work file is empty and has the character set *NONE then it is assigned the character set EDF041 prior to insertion.

If the information is inserted in a string variable then it is converted into the character set of the string variable before insertion.

In this statement, the statement name may be omitted entirely. In F mode, it is also permissible to omit the statement symbol.

Example

```
1.     @SET #L0 = 1 ------------------------------------------------- (01)
1.     @SET #I0 = 123 ----------------------------------------------- (02)
1.     @SET #L0 = CHAR #I0 ------------------------------------------ (03)
1.     @SET #L0 ,13 = CHAR #S0 -------------------------------------- (04)
1.     @SET #L0 ,18 = CHAR #L0 -------------------------------------- (05)
1.     @SET #S0 = CHAR #I0 ------------------------------------------ (06)
1.     @SET #L0 = 47.11 --------------------------------------------- (07)
1.     @SET #S1 = CHAR #L0 ------------------------------------------ (08)
1.     @SET #S2 ,5 = CHAR #S0 --------------------------------------- (09)
1.     @PRINT 1,#S0-#S2 --------------------------------------------- (10)
1.0000  0000000123 #S00    1.0000
  #S00  0000000123
  #S01   47.1100
  #S02 #S00
```

(01) The value `0001.0000` is assigned to the line number variable `#L0`.

(02) The value `123` is assigned to the integer variable `#I0`.

(03) The string `' 0000000123'` is inserted in line 1 as of column 1.

(04) The string `'#S00'` is inserted in line 1 as of column 13.

(05) The string `'   1.0000'` is inserted in line 1 as of column 18.

(06) The value `' 0000000123'` is assigned to the string variable `#S0`.

(07) The value `47.11` is assigned to the line number variable `#L0`.

(08) The value `' 47.1100'` is assigned to the string variable `#S1`.

(09) The value `'#S00'` is assigned to the string variable `#S2` starting at column 5, i.e. its value is `'    #S00'`.

(10) Line 1 and the string variables `#S0..#S2` are output.

9.113 @SET (format 5) – Date and time

This format of the @SET statement is used to store the date and time in a string variable or a work file. The value is stored starting at a specified column.

Operation	Operands	F mode, L mode
@SET	{ svarex / lvar } [,col] = { **D**ATE [**I**SO[**4**]] / **T**IME }	

svarex — String variable (#S0..#S20) in which the date or time are to be inserted. Any characters in the corresponding positions are overwritten.

lvar — Line number variable (#L0..#L20) specifying the line in which the date or time are to be inserted. If the line does not yet exist, it is created. Any characters in the corresponding positions are overwritten.

col — Column as of which the date or time are to be stored. If col is not specified then the values are inserted starting at column 1. If col is located after the end of the line then the line is filled with blanks up to the column col.

DATE — The current date is inserted in the specified string variable or line in the desired form. If ISO is not specified then the form mm/dd/yyjjj is used. Here, mm specifies the month, dd the day, yy the year and jjj the day of the year.

ISO — Specifies that the date is to be output in the format yy-mm-ddjjj.

ISO4 — Specifies that the date is to be output in the format yyyy-mm-ddjjj.

TIME — The time is stored in the specified string variable or line in the form hhmmss. Here, hh specifies the hours, mm the minutes and ss the seconds.

If the insertion operation causes the line or string variable to exceed the maximum length of 32768 then the statement is rejected with the message EDT5474.

If the information is inserted in a line then the character set depends on the work file. If the current work file already has a character set then the value is inserted in this character set. If the work file is empty and has the character set *NONE then it is assigned the character set EDF041 prior to insertion.

If the information is inserted in a string variable then it is converted into the character set of the string variable before insertion.

In this statement, the statement name may be omitted entirely. In F mode, it is also permissible to omit the statement symbol.

Example

```
1.     @SET #L0 = 1 ------------------------------------------------- (1)
1.     @SET #L0 = DATE ---------------------------------------------- (2)
1.     @SET #L0 ,13 = DATE ISO -------------------------------------- (3)
1.     @SET #S0 = TIME ---------------------------------------------- (4)
1.     @SET #S0 ,13 = DATE ISO4 ------------------------------------- (5)
1.     @PRINT 1 ----------------------------------------------------- (6)
1.0000 02/07/06038 06-02-07038
1.     @PRINT #S00 -------------------------------------------------- (7)
  #S00 165941      2006-02-07038
```

(1) The value `0001.0000` is assigned to the line number variable `#L0`.

(2) The date is inserted in line 1 as of column 1.

(3) The date is inserted in ISO format in line 1 as of column 13.

(4) The time is inserted in the string variable `#S00` starting at column 1.

(5) The date in `ISO4` format is inserted in the string variable `#S00` starting at column 13.

(6) Line 1 is output.

(7) The string variable `#S00` is output.

9.114 @SET (format 6) – Modify current increment and line number

This format of the @SET statement defines the current line number and the current increment or restores earlier values for the line number and increment.

Operation	Operands	F mode, L mode
@SET	[line [(inc)] [:text]]]	

line — This operand specifies the new current line number.

inc — This operand specifies the new current increment. If `inc` is not specified then the increment implicitly specified by `line` is used (see section "Implicit increment assignment" on page 35).

text — EDT statement or data input which is executed or inserted in the new current line after the new current line number and increment have been defined. The string is treated as if it had been entered at the prompt in L mode. In particular, the decision to interpret the text as data input or as a statement is made in accordance with the same rules (for more information, see section "L mode" on page 126).

The `text` operand starts immediately after the character `':'`, i.e. any specified blanks form part of the operand and are taken over into the line in the case of data input.

If `text` is not specified (although the colon is), then an empty line (line of length 0) is inserted. If neither the `text` operand nor the colon are specified then only the new current line and increment are defined.

Every @SET line [(inc)] statement defines a new current line number and a new current increment. The values that previously represented the current line number and increment for the current work file are stored in a memory area which can accommodate a maximum of three pairs of values (line number/increment).

The pairs of values previously stored in this memory area slip down one place every time the values are redefined. When the memory area is full the last (oldest) pair of values is lost.

If the @SET statement is specified without operands then the last pair of values saved in this memory area is used as the current line number and current increment.
The movement of the pairs of values in the memory area depends on whether or not the memory area was already full. If the memory area was not already full then all the pairs of values move forward one place.
The last place in the memory area remains empty. If the memory area is already empty when @SET is issued without operands then the previous current line number and the previous current increment are retained and the message `EDT4964` is output.

If the memory area is already full when the @SET statement is specified without operands then the last pair of values saved in this memory area is again used as the current line number and current increment.

However, in this case, the values in the memory area are rotated, i.e. all the pairs of values in the memory area except for the first pair move forwards one place and what was previously the first pair in the memory area (and now forms the new current line number and the new current increment) is stored in the last place in the memory area.

If the work file is completely deleted then the memory area is emptied.

In this statement, the statement name may be omitted entirely. Unlike the other formats of the @SET statement, in F mode it is then necessary to specify the statement symbol.

Particular attention is required when using the `text` operand in F mode. When it is used, EDT temporarily switches to L mode and then reactivates F mode again.
This also occurs if the operand contains a statement which causes a switch to L mode. If the operand contains data input which in turn contains a semicolon or non-paired double quotes then this results in the statement line being broken down into subsegments, possibly with unexpected consequences.

9.115 @SETF – Change work file and set position

The @SETF statement sets the vertical and horizontal position of the work window for a work file either with or without changing the current work file.

Operation	Operands	F mode, L mode
@SETF **@#**	[{$0..$22 | **GL**OBAL | (0..22)}] [{line | vpos}] [{:col: | hpos}]	

$0..$22 Work file in which the position is to be set. The current work file remains unchanged. If no further operands are specified then the position in the specified work file is set to the (logical) first line and first column.

0..22 Work file in which the position is to be set. The specified work file is set as the current work file before positioning. Before EDT goes to the specified work file any nesting of the work files undertaken with @PROC is undone. If no further operands are specified then the specified work file is simply set as the current work file and the line and column positions in the specified work file remain unchanged.

An *active* work file cannot be specified as the current work file. If the user attempts to do so, the message `EDT4959` is issued.

GLOBAL Positioning is performed simultaneously in all the work files. If no further operands are specified then the position is set to the (logical) first line and first column in all the work files.

line Absolute specification of the vertical position. The user must specify the line number which is to be displayed in the first line of the work window. If no line with this line number is present then the position is set to an existing line which has the next (higher) line number. If there is no such line then the position is set to the existing line with the highest line number.

vpos Relative vertical positioning statement. It is possible to specify `+[n]`, `++`, `-[n]`, `--` as well as `+([m[,...]])`, `++([m[,...]])`, `-([m[,...]])` and `--([m[,...]])`.

Here, `m` is one of the 9 possible record marks to which the position may be moved. Multiple record marks can be specified.

Marks with special functions (e.g. mark 15 for write protection, see section “Record marks” on page 45) are ignored here.

The relative positioning statement acts in the same way as the corresponding statement (+[n], ++ etc.) in F mode if this is transferred with DUE (see the corresponding statement). However, positioning with @SETF is also possible in L mode (for example in procedures) or when EDT is controlled via the subroutine interface.

col — Absolute specification of the horizontal position. The user must specify the number of the column which is to be displayed as the first column in the work window.

hpos — Relative horizontal positioning statement. It is possible to specify >[n], <[n] and <<.

The relative positioning statement acts in the same way as the corresponding statement (>[n], << etc.) in F mode if this is transferred with DUE (see the corresponding description). However, positioning with @SETF is also possible in L mode (for example in procedures) or when EDT is controlled via the subroutine interface.

If @SETF is specified without any operands then the position in the current work file is set to the (logical) first line and first column. If @SETF is specified without positioning operands then @SETF (n) simply changes the current work file. All the positions are retained. If @SETF $n or @SETF GLOBAL is specified then the position in the relevant work files is set to the (logical) first line and first column.

The abbreviation # may only be entered in F mode. At least one operand must be specified. Otherwise, the # statement (output last statement) is executed.

The setting for the window position is saved separately for the upper and lower (possible) data windows corresponding to each work file. The effect of the set position depends on the visibility of the affected window and possibly on the current input window. This applies equivalently for input in L mode where *visibility* and *input window* refer to a possible switch to F mode (with the @EDIT FULL statement). The corresponding information can be displayed in L mode using @STATUS=PAR(..).

In the case of non-visible work files, the position is set for the upper window.

If an affected work file is visible in precisely one window (upper or lower) then the positioning instructions apply to this window irrespective of which window is the input window.

If a work file is visible in both windows when the screen is split then only the input window is positioned.

Note

If @SETF is used to make settings for work files (GLOBAL or $0..$22 operand), these no longer temporarily become the current work file as they do in compatibility mode. The restrictions and side effects described there do not therefore occur in Unicode mode.

Example

This example assumes that work file 1 already contains records and that @PAR LOWER=ON is set.

```
  23.00 ·····························································
@setf (1) 4 :3:·····························································0000.00:00001(02)
```

EDT is to switch to work file 1 and set the position there to line number 4, column 3.

```
   4.00 iendly creation and editing<·····································
   5.00 BS2000 files in SAM and ISAM formats<····························
   6.00  s well as text-like library<····································
   7.00 ements and POSIX files<·····································
:                                                                      :
:                                                                      :
:                                                                      :
·····························································0004.00:00003(01)
```

The switch to work file 1 and the positioning operation have been performed.

9.116 @SETJV – Catalog job variable and assign value

The @SETJV statement enters a job variable in the catalog and assigns it a value.

Operation	Operands	F mode, L mode
@SETJV	{[string] = string1[,...] [,**C**ODE=name] / string}	

string — String which specifies the fully qualified name of a job variable. Although the name must comply with the syntactic rules for job variable names, EDT does not check these rules in full.

If the job variable is not yet present in the catalog, it is cataloged using the standard functions of the `DCLJV` macro.

The link name `*EDTLINK` is assigned to the job variable. This can be used to address it in subsequent statements. The assignment is performed before the corresponding system interfaces are called. If errors then occur when values are assigned, the assignment that has already been performed is retained.

If `string` is not specified then the job variable is addressed using the link name `*EDTLINK`. In this case, `string1` must be specified. If no job variable is linked to the link name `*EDTLINK` then the statement is rejected with the message `EDT5289`. If `string1` is also not specified then `the statement` is rejected with the message `EDT3908`.

If the job variable cannot be accessed then the statement is aborted with the message `EDT4208`.

string1 — One or more strings that are to be assigned to the job variable.

If multiple strings are assigned then they are chained together to form an intermediate result. If all the strings involved have the same character set then this is also the character set of the intermediate result. If the involved strings have different character sets then the character set of the intermediate result is `UTFE`.

If the string is longer than 256 bytes then only the first 256 bytes are assigned as a value and the message `EDT1936` is output.

If `string1` is not specified or if `string1` is an empty string then an empty job variable is created. If it already exists, its value is not changed.

name — Name of a character set into which the intermediate result is converted before being assigned to the job variable. If `name` is not specified, `EDF041` is used. The character set name must be known in XHCS; otherwise, the statement is rejected with message `EDT4980`.

If the string that is to be assigned contains characters which are invalid in the specified character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, no assignment is performed and the error message `EDT5453` is output.

If the Job Variable Support subsystem is not installed, the statement is rejected with the error message `EDT5254`. For details concerning job variables, see the User Guide JV [9].

9.117 @SETLIST – Extend list variable

The @SETLIST statement assigns elements to an S list variable. In this case, values are taken over from lines in the current work file or from string variables.

If no values are specified then the content of the S list variable is deleted.

Operation	Operands	F mode, L mode
@SETLIST	string { [lines[,...]] [**MA**RK [m]] / svar } [:cols[,...]:] [,] [**M**ODE= { **A**PPEND / **P**REFIX / **O**VERWRITE }] [,**C**ODE = name]	

string — String which specifies the valid name of an S list variable. Although the name must comply with the syntactic rules for S variable names, EDT does not check these rules in full.

lines — One or more line ranges whose contents are to be taken over into the S list variable. If no lines are present in the specified line range then the message `EDT2903` is output. If no line range is specified then all the lines in the current work file are used.

MARK — Only marked lines in the specified line ranges are to be taken over. If `MARK` is not specified then all the lines are taken over.

m — Number of the record mark (`1..9`) that is to be used as a selection criterion. If `m` is not specified, then only data lines with record mark 1 are used.

svar — Name of a string variable whose content is to taken over as an element.

cols — One or more column ranges whose characters are to be taken over. The ranges may repeat and overlap.
If a line in a column that is to be used does not contain any characters then a blank is inserted instead of it.
All the specified characters are concatenated in the sequence in which the columns are specified (possibly multiple times) and the result is inserted in the list element.

If the operand is not specified, the entire line is taken over.

MODE= Specifies the way in which a list is to be extended.

APPEND The list is extended at the end, i.e. the new list elements are appended after the last element (default value).

PREFIX The list is extended at the start, i.e. the new list elements are inserted before the first element.

OVERWRITE
The content of the S list variable is first deleted. The new list elements are then taken over.

If no line is present in the specified line range then the S list variable is not assigned any further elements. In F mode, the message EDT0211 informs the user of this.

name Character set in which the string for assignment is to be converted before being assigned. If name is not specified, EDF041 is used. The character set name must be known in XHCS; otherwise, the statement is rejected with message EDT4980.

The comma in front of the MODE operand must be specified in order to distinguish it from MARK if no other operand is specified apart from the list name or any range specification.

The list variable must already exist. If it does not, the message EDT5274 is issued. If it is not a list variable, the message EDT4910 is output. If the list variable is not of type STRING or ANY then the statement is rejected with the message EDT5343.

If an S variable is extended by means of APPEND or PREFIX then, in F mode, the message EDT0210 is output to inform the user of this.

In L mode, if the work file is empty and a line range (not a string variable) has been specified then the message EDT2903 is output.

If the string that is to be assigned contains characters which are invalid in the specified character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, no assignment is performed and statement is aborted with the error message EDT5453.

If the value that is to be assigned is longer than 4096 bytes then only the first 4096 bytes are assigned as a value and the message EDT2403 is output.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

For details on S list variables, see the SDF-P User Guide [7].

9.118 @SETSW – Set job and user switches

The @SETSW statement is used to set or reset user and job switches.

Operation	Operands	F mode, L mode
@SETSW	[<u>**ON**</u> / **OF** F] =] {[**U**] int1[-int2]} [,...]	

ON — The specified switches are set (default value).

OFF — The specified switches are reset.

U — If `U` is specified then the subsequent input of `int1` and possibly of `int2` applies to a user switch under the user's own ID.

If `U` is not specified then the subsequent input of `int1` and possibly of `int2` applies to a job switch.

int1 — Number of the switch (`0..31`) that is to be set or reset.

int2 — All the switches between `int1` and `int2 (0..31)` are set or reset. If `int2` is smaller than `int1`, then the statement is rejected with the message `EDT3216`.

Format 4 of the @IF statement can be used to check whether or not a job switch or user switch is set for the user's own ID.

In a @SETSW statement, it is possible to set or reset both user switches and job switches.

Example 1

```
@SET #I2 = 6
@SETSW ON = U1-#I2,12-20,U31
```

User switches 1 to 6 and 31 and job 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 ---------------------------------------- (1)
   2.     @ @PRINT #S2 N
   3.     @ @RETURN
   4.     @ @PRINT #S3 N
   5.     @END
   1.     @SETSW OFF = 15 ------------------------------------------------ (2)
   1.     @DO 9 ---------------------------------------------------------- (3)
SWITCH 15 IS OFF
   1.     @SETSW ON = 15 ------------------------------------------------- (4)
   1.     @DO 9 ---------------------------------------------------------- (5)
SWITCH 15 IS ON
   1.
```

(1) A procedure is stored in work file 9 that outputs the string variable #S3 if job switch 15 is set, and the string variable #S2 otherwise.

(2) Job switch 15 is reset.

(3) The procedure in work file 9 is executed.

(4) Job switch 15 is set.

(5) The procedure in work file 9 is executed.

9.119 @SETVAR – Declare S variable and assign value

The @SETVAR statement is used to declare an S variable and/or assign a value to an S variable.

Operation	Operands	F mode, L mode
@SETVAR	string [= string1 | ivar] | **SYSEDT** [,**K**EEP] [,**M**ODE= **A**NY | **N**EW | **U**PDATE] [,**C**ODE = name]	

string — String which specifies a valid S variable name. Although the name must comply with the syntactic rules for S variable names, EDT does not check these rules in full.

string1 — String which is to be assigned to the S variable.

If the S variable is not of type `STRING` or `ANY` or if it is an array or list then the statement is aborted with the message `EDT5342`.

If the value that is to be assigned is longer than 4096 bytes then only the first 4096 bytes are assigned as a value and the message `EDT2403` is output.

ivar — Integer variable (`#I0..#I20`) whose content is to be assigned as a value to the S variable specified in `string`.

If the S variable is not of type `INTEGER` or `ANY` or if it is an array or list then the statement is aborted with the message `EDT5342`.

SYSEDT — The contents of the string variables `#S00` to `#S20` are assigned to the S variables `SYSEDT-S00..SYSEDT-S20`. If errors occur, the corresponding messages are output and the same message may be repeated. This does not terminate the statement.

In the case of S variables to which no string can be assigned as a value (other type) no error is reported and no assignment is performed. The treatment of non-existent S variables depends on the `MODE` setting.

KEEP If KEEP is specified then EDT is set not to overwrite the S variables SYSEDT-S00..SYSEDT-S20 on termination. If KEEP is not specified then the setting is canceled, i.e. on termination, EDT assigns the contents of the string variables #S00 to #S20 using the character set EDF041.

MODE= Specifies whether the S variable should already exist.

- ANY A value is assigned to an existing or a new S variable.
- NEW The S variable must not already exist. If it already exists then the statement is not executed and the message EDT5272 is output. If SYSEDT is specified then the specification of NEW is treated in the same way as ANY.
- UPDATE The S variable must already exist. If it does not already exist then the statement is not executed and the message EDT5274 is output. If SYSEDT is specified then the message is not output and values are only assigned to existing S variables of type STRING or ANY.

name Character set in which the string for assignment is to be converted before being assigned. If name is not specified, EDF041 is used. If string is specified but string1 is not then the operand is ignored. Otherwise, the character set name must be known in XHCS. If it is not, the statement is rejected with message EDT4980.

If string is specified but there is no specification for either string1 or ivar then an empty string is assigned as a value to the S variable. If it does not yet exist then it is created with default attributes (TYPE=ANY,MULTIPLE-ELEMENTS=*NO,SCOPE=PROCEDURE). If the variable already exists and has incompatible attributes then the statement is rejected with the message EDT5342.

If the string that is to be assigned contains characters which are invalid in the specified character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, no assignment is performed and the error message EDT5453 is output.

For details on S variables, see the SDF User Guide [6].

9.120 @SHIH – Output statement buffer

The @SHIH statement is used to output the EDT statement buffer. Only statements input in F mode are entered in the statement buffer.
The scrolling statements, the statements for changing the work file and the @SHIH itself are not entered in the statement buffer.

Operation	Operands	F mode, L mode
@SHIH	[[**T**O] line [(inc)]] [**F**ORWARD]	

line — Line number as of which information is to be written to the current work file.

If a line with a number greater than the previous highest line number is created then the current line number is modified.

If `line` is not specified then in the interactive mode's L mode, the result is output to `SYSOUT` (in batch mode, no output is possible) and in F mode it is written to work file `9`. Work file `9` is deleted before being used. If a file is open in work file `9` then the message `EDT5189` is output and the statement is not executed.

inc — Increment used to form the line numbers which follow `line`. If `inc` is not specified then the increment implicitly specified by `line` is used (see section “Implicit increment assignment” on page 35).

FORWARD — If this operand is specified then the statements are output in the sequence in which they were entered. If it is not specified then they are output in the opposite sequence, i.e. the last entered statement is output first.

The statement buffer can accommodate a maximum of 2048 statements independently of their respective lengths. If the statement buffer is empty then the @SHIH statement is rejected with the message `EDT5376`.

This operation takes no account of whether a statement was entered in the upper or lower part of a split screen. Statements are stored in the statement buffer independently of the work file to which the statement was applied. Statements in a statement sequence (statements separated by ’;’) are stored individually.

If output is written to work file `9` (in F mode, without the line operand) then the header is output in the information line (can be displayed using @PAR INFORMATION=ON). In addition, a @LOWER ON statement is implicitly issued for work file 9.

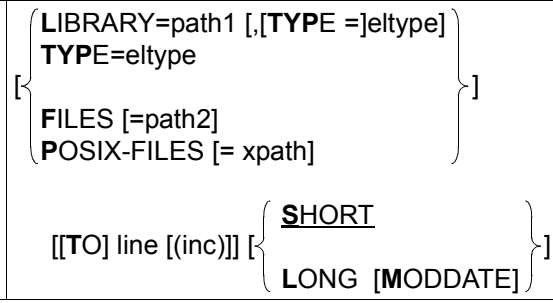
Output to SYSOUT is sent in the character set that has been defined for this system file. If the output is written to a work file then it is sent in the work file's character set. If the work file is empty and has the character set *NONE then the character set UTFE is used. Characters that cannot be displayed in the target character set are always replaced by blanks.

Note

The statement code K can be used to copy a statement to the statement line. This can then be executed in another work file if it is preceded by a statement that changes work file (see the $0..$22 statement).

9.121 @SHOW (format 1) – Output directory

The @SHOW statement (format 1) can be used to output a library's directory or a list of files from the BS2000 catalog or from a POSIX directory. It is possible to define the destination for the output. Optionally, it is also possible to output additional information about the files or library elements.

Operation	Operands	F mode, L mode
@SHOW		

LIBRARY= The directory of a library or of an element type in a library is to be output. If there is more than one version of an element then all the versions are displayed. If there are no elements of the specified element type, the message EDT5287 is output. If the specified library does not exist or cannot be accessed as required then a corresponding message is output.

If LIBRARY (and also FILES and POSIX-FILES) is not specified then the default library set with @PAR LIBRARY is used implicitly provided that @PAR LIBRARY has been specified. Otherwise, the message EDT5181 is output.

path1 Name of the library.

TYPE= Only the directory of all the elements of a specific type should be specified. If TYPE is not specified then the entire library directory is output.

eltype Type of element. Permitted type specifications are S, M, P, J, D, X, R, C, H, L, U, F, *STD and freely selectable type names having one of these types as basic type. The permitted element types and their meanings are described in chapter “File processing” on page 131.

FILES= A list of files from the BS2000 catalog is to be output.

path2 Designates the files that are to be listed. The `path2` operand can be a fully or partially qualified file name, may contain wildcards and can be up to 80 characters in length.

If `path2` is not specified then a list of all the files under the user's own ID is output. If no file with the specified name is found, the message `EDT5281` is output.

POSIX-FILES=
A list of files from the POSIX directory is to be output.

xpath Designates the POSIX files that are to be listed. If `xpath` is a directory then all the files in this directory are listed (without the directory component). If `xpath` is an ordinary file then its name is displayed as specified.

The `xpath` operand can also be specified as a string variable. It *must* be specified as a string variable if the path name contains characters which have a special meaning in EDT syntax (e.g. blanks, semicolons in F mode or commas).

If `xpath` is not specified then a list of all the files in the current POSIX directory is output. If no file with the specified name is found or if the directory cannot be accessed as required then a corresponding message is output.

line Line number as of which information is to be written to the current work file.

If a line with a number greater than the previous highest line number is created then the current line number is modified.

If `line` is not specified then in the interactive mode's L mode, the result is output to `SYSOUT`, in batch mode it is output to `SYSLST` and in F mode it is written to work file 9. Work file 9 is deleted before being used. If a file is open in work file 9 then the message `EDT5189` is output and the statement is not executed.

inc Increment used to form the line numbers which follow `line`. If `inc` is not specified then the increment implicitly specified by `line` is used (see section “Implicit increment assignment” on page 35).

SHORT This is the default value for handling information output. The meaning of SHORT is different for the individual file types.

In the case of libraries, every element is output:

Column	Header	Meaning
2–5	TYP	Element type
7–38	ELEMENT	Element name
42–53	VERSION	Version designation or @ for the highest possible version
56–59	VAR	Variant number
63–72	DATE	User date (format YYYY-MM-DD)

The list is alphabetically sorted on the type names, element names and version names. In the case of type names > 4, element names > 32 or version designations > 12 characters, entries consist of 2 lines.

If the output is sent to SYSOUT or SYSLST then it is accompanied by a header (see table). If the output is sent to work file 9 (in F mode, without the line operand) then the header is output in the information line (can be displayed using @PAR INFORMATION=ON). If the output is written to a work file due to the line operand then no header is output.

In the case of BS2000 files, one file is output per line. Only the extended file names together with the catalog ID and user ID, are output. The list is alphabetically sorted on the file names.
No header is displayed.

In the case of POSIX files, one file is output per line. Only the file names are output. The list is alphabetically sorted on the file names. No header is displayed.

LONG Additional information is output for the files. The meaning of LONG is different for the individual file types.

In the case of libraries, a line with the following content is output for every element:

Column	Header	Meaning
1-8	TYP	Element type
10-73	ELEMENT L=path1	Element name
75-98	VERSION	Version designation or @ for the highest possible version
100-103	VAR	Variant number
105-114	DATE	User date or date of last modification (format YYYY-MM-DD)
116-123	CODESET	Character set

The list is alphabetically sorted on the type names, element names and version names.

If the output is sent to SYSOUT or SYSLST then it is accompanied by a header (see table) which includes the name of the library. If output is written to work file 9 (in F mode, without the line operand) then the header is output in the information line (can be displayed using @PAR INFORMATION=ON). If the output is written to a work file due to the line operand then no header is output.

In the case of BS2000 files, a line with the following content is output for every file:

Column	Header	Meaning
1-10	SIZE	Number of PAM pages
11	P	File on private or public data medium (*/␣)
12-65	FILENAME	File name with CATID and USERID
67-76	LAST PP	Last used PAM page
78-87	CR-DATE or MOD-DATE	Creation date or date of the last modification (format YYYY-MM-DD)
89	S	SHARE attribute (Y/N/S)
90	A	ACCESS attribute (W/R)
92-95	FCB	FCB type (SAM/ISAM/PAM/BTAM/NONE)
97	R	READ-PASS attribute (Y/N)
98	W	WRITE-PASS attribute (Y/N)

Column	Header	Meaning
100-107	CODESET	Character set

The list is alphabetically sorted on the file names.

If the output is sent to SYSOUT or SYSLST then it is accompanied by a header (see table). If the output is sent to work file 9 (in F mode, without the line operand) then the header is output in the information line (can be displayed using @PAR INFORMATION=ON). If the output is written to a work file due to the line operand then no header is output.

In the case of POSIX files, a line with the following content is output for every file:

Column	Header	Meaning
1	T	Type (D for directory, L for symbolic link, B for device file (block), C for device file (character), M for file with distributed access, P for FIFO, S for semaphore file or F for file)
2-10	ACCESS	Access rights (RWX for *user*, *group*, *others*)
12-31	SIZE	Size of file in bytes
33-42	MOD-DATE	Date of last modification (format YYYY-MM-DD)
44+	FILENAME	File name (case-sensitive)

The list is alphabetically sorted on the file names.

If the output is sent to SYSOUT or SYSLST then it is accompanied by a header. If the output is sent to work file 9 (in F mode, without the line operand) then the header is output in the information line (can be displayed using @PAR INFORMATION=ON). If the output is written to a work file due to the line operand then no header is output.

MODDATE In the case of BS2000 files, the date of the last modification is output instead of the creation date. The date format is unchanged. MOD-DATE is output in the header.
In the case of libraries, the date of the last modification is output instead of the user date.

In the case of POSIX files, this operand is ignored.

Output to SYSOUT or SYSLST is sent in the character set that has been defined for these system files.
If the output is written to a work file then it is sent in the work file's character set. If the work file is empty and has the character set *NONE then the character set EDF041 is used. Characters that cannot be displayed in the target character set are always replaced by blanks.

Example

```
   1.00 @SHOW DISPLAYS THE DIRECTORY FOR A LIBRARY.<···························
   2.00 THE NAME OF THE LIBRARY MUST BE SPECIFIED IN THE OPERAND..<·············
   3.00 ·······································································
show library=edt.lib.xmpl to 100(10)·····························0001.00:00001(00)
```

The entire directory of the library EDT.LIB.XMPL is to be output in the current work file starting at line number 100 and using the increment 10.

```
   1.00 @SHOW DISPLAYS THE DIRECTORY FOR A LIBRARY.<···························
   2.00 THE NAME OF THE LIBRARY MUST BE SPECIFIED IN THE OPERAND.<·············
 100.00  C    PROG                                @             0003   2006-01-11
 110.00  C    PROG1                               @             0004   2006-01-11
 120.00  D    E.TEXT1                             @             0006   2005-12-05
 130.00  D    E.TEXT2                             @             0013   2005-12-05
 140.00  D    E.TEXT3                             100           0011   2003-12-13
 150.00  D    E.TEXT3                             101           0121   2003-12-20
 160.00  D    E.TEXT3                             102           0007   2004-04-11
 170.00  J    ASSEMB                              @             0099   2006-01-11
 180.00  J    FILE-TRANSFER                       @             0045   2006-01-11
 190.00  J    PROC1                               1             0001   2005-12-05
 200.00  D    THIS-IS-AN-ELEMENT-WITH-A-VERY-L    @             0000   2005-08-17
 210.00       ONG NAME
 220.00  D    THIS-IS-AN-ELEMENT-WITH-A-VERY-L    VERY-OLD.VER  0000   2003-08-17
 230.00       ONG-VERSION                         SION
 240.00 FREE THIS-IS-AN-ELEMENT-WITH-A-FREE-T     @             0000   2003-08-17
 250.00 TYPX YPE-NAME
 251.00 ·····································································
 252.00 ·····································································
 253.00 ·····································································
 254.00 ·····································································
 255.00 ·····································································
show files=*text* long;edit long·····························0001.00:00001(00)
```

Detailed information on all the BS2000 files whose names contain the string TEXT is output in work file 9. EDIT LONG is specified to make it possible to view the information in full in the data window.

```
0000000003 :N:$USER.XMPL.TEXT                                  0000000002 200
7-01-12 NW ISAM NN *NONE   <·····································
0000000006 :N:$USER.TEXT.PROG                                  0000000006 200
7-01-12 YR PAM  NY *NONE   <·····································
0000000015 :N:$USER.TEXT1                                      0000000014 200
7-01-12 NW SAM  NN EDF041  <·····································
0000000021 :N:$USER.TEXT2                                      0000000020 200
7-01-12 NW SAM  NN EDF03IRV<·····································




show posix-files=data long;index on·····························0001.00:00001(09)
```

Detailed information about the POSIX file “data” in the current POSIX directory is output in work file 9. EDIT-LONG mode is deactivated again.

```
   1.00 FRW-RW-RW-  00000000000000000004  2006-11-08  data <·····················
   2.00 ·····································································




·····························································0001.00:00001(09)
```

9.122 @SHOW (format 2) – Output supported character sets

The @SHOW statement (format 2) can be used to output a list of the character sets supported by XHCS. In interactive mode, it also indicates the character sets supported by the terminal.

Operation	Operands	F mode, L mode
@SHOW	**C**CS [[**T**O] line [(inc)]]	

line — Line number as of which information is to be written to the current work file.

If a line with a number greater than the previous highest line number is created then the current line number is modified.

If `line` is not specified then in the interactive mode's L mode, the result is output to `SYSOUT`, in batch mode it is output to `SYSLST` and in F mode it is written to work file 9. Work file 9 is deleted before being used. If a file is open in work file 9 then the message `EDT5189` is output and the statement is not executed.

inc — Increment used to form the line numbers which follow `line`. If `inc` is not specified then the increment implicitly specified by `line` is used (see section “Implicit increment assignment” on page 35).

If output is written to work file 9 (in F mode, without the `line` operand) then a header is output in the information line (can be displayed using @PAR INFORMATION=ON).

A list of the character sets supported by XHCS is output. Unlike the behavior in EDT V16.6 or in compatibility mode, in Unicode mode each of the listed character sets can be defined as the character set for a work file.

Alongside the name of the character set, every line in the list contains an additional indicator, namely:

`P` If the character set is a partial character set (see the XHCS User Guide [8]).
`E` If the character set is an EBCDIC character set.
`I` If the character set is an ISO character set.
`*` If the character set is one of the character sets accepted by the terminal (in interactive mode only).
The character sets identified by `*` are also those that can be defined as the communications character set using @CODENAME (format 2).

Output to SYSOUT or SYSLST is sent in the character set that has been defined for these system files. If the output is written to a work file then it is sent in the work file's character set. If the work file is empty and has the character set *NONE then the character set EDF041 is used. Characters that cannot be displayed in the target character set are always replaced by blanks.

9.123 @SORT – Sort line ranges

The @SORT statement is used to sort contiguous line ranges in the current work file in ascending or descending order. By specifying a column range, it is possible to restrict the sort operation to the relevant section of the record.

Operation	Operands	F mode, L mode
@SORT	[lines] [-{ :cols[:] / :**R** (cols*) }] [-{ **A** / D }]	

lines — Line range in which the data is to be sorted. If no line range is specified then all the records in the current work file are sorted.

cols — Column range whose characters are to be included in the sort operation. This column range is referred to as the *sort field* below.

If only one column number is specified then the characters from this column up to the end of the line are included in the operation. If the first column specification is greater than the line length then the corresponding line is treated as if it contained an empty sort field.

The sort fields in two lines are compared one character at a time from left to right. If the end of one of the sort fields is reached during this comparison (with no differences being discovered) then the line with the shorter sort field is considered to be smaller.

If no column range is specified then the sort field comprises the entire line.

cols* — Column range starting at the end of the record. Counting from the end of the record back toward the start of the record simply determines the number of characters present in the sort field. Despite this, the characters in the sort field are compared from left to right. The specification `1-10`, for example, therefore means the last ten characters in every line.

A — Sorting is performed in *ascending* order.

D — Sorting is performed in *descending* order.

Lines with identical sort fields retain their original sequence in the work file.

If a 7-bit or 8-bit character set is assigned to the work file then the sequence of the individual characters is determined by their byte codes interpreted as binary numbers. If a Unicode character set is assigned to the work file then the sequence of the individual characters is determined by the UTF16 codes of the characters interpreted as binary numbers. The sort weighting managed by XHCS for the individual characters is ignored. The sort sequence when sorting with EDT may therefore differ from the sort sequence when sorting with the SORT program.

The @SORT statement uses a combination of *quicksort* and *bubblesort*.

The @SORT statement cannot be used for a file opened for real processing with @OPEN (format 2).

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Examples

```
@SORT :1-15
```

Sorts all the records in the current work file in ascending order on the basis of the contents of columns 1 to 15.

```
@SORT &:1-15
```

Sorts all the records in the range defined using the range symbol & (see @RANGE statement) in the current work file in ascending order on the basis of the contents of columns 1 to 15.

```
@SORT %-.%+19L :R(1-8) D
```

Sorts the first 20 records in the current work file in descending order on the basis of the contents of the last 8 columns.

```
@SORT 20-.$:#I1-#I2
```

Sorts the records from line number 20 through to the end of the current work file in ascending order. The column range within which sorting is performed is defined by the integer variables #I1 and #I2.

9.124 @SPLIT – Display 2 work windows

In F mode, the @SPLIT statement can be used to activate or deactivate the display of a second work window on the screen. Each work window has a separate statement line (see also section "The work window" on page 103).

Operation	Operands	F mode
@SPLIT	{ n { (0..22) | $0..$22 } | **O**FF }	

n — Specifies the number of lines for the lower work window including the statement line. A value of 2 or more must be specified for n and this value must not be more than two less than the maximum number of lines permitted by the terminal for the screen format in question (see @VDT statement).

0..22 | $0..$22 — Number of the work file that is to be displayed in the lower work window. The upper work window contains the work file in which the statement was issued. Users are permitted to define the same work file for both work windows. This makes it possible, for example, to edit different line ranges in the same work file at the same time.

OFF — The work window containing the statement line in which the statement was issued is displayed in full. The specification of 0 (Null) instead of OFF is now only supported for reasons of compatibility.

When an EDT session starts, the work window is not split in F mode.

The cursor is positioned in the upper statement line once the screen has been split. After each subsequent output, it is positioned in the statement line in which the last statement or statement sequence was entered. If a statement is entered in both statement lines, then the cursor is positioned in the upper statement line after the statements have been executed. If an error occurs while processing a statement then the cursor is positioned in the statement line in which the invalid statement was entered.

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

The @PAR SPLIT statement can be used instead of @SPLIT and has the same functionality. Furthermore, @PAR SPLIT can be used for a specific work file or globally for all the work files and is also permitted in L mode and therefore in EDT procedures.

Example

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
   2.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   3.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································································
:                                                                             :
:                                                                             :
:                                                                             :
@split 10(2)·····················································0001.00:00001(00)
```

@SPLIT 10(2) requests a second work window which contains 10 lines including the statement line. Work file 2 is to be displayed in this work window.

```
   1.00 BERGER   ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····················
   2.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   3.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································································
   7.00 ···············································································
   8.00 ···············································································
   9.00 ···············································································
  10.00 ···············································································
  11.00 ···············································································
  12.00 ···············································································
  13.00 ···············································································
·····························································0001.00:00001(00)
   1.00 YOU CAN NOW<···················································
   2.00 EDIT WORK FILES 0 AND 2<·····················································
   3.00 IN ALTERNATION OR<·····························································
   4.00 SIMULTANEOUSLY<··································································
   5.00 ···············································································
   6.00 ···············································································
   7.00 ···············································································
   8.00 ···············································································
   9.00 ···············································································
·····························································0001.00:00001(02)
```

9.125 @STAJV – Output job variable information

The @STAJV statement outputs information about job variables or writes this information to a work file.

Operation	Operands	F mode, L mode
@STAJV	[string] [**T**O line [(inc)]] [{ **S**HORT] / **LONG** [ISO**4**] }]	

string — String indicating the names of the job variables whose attributes are to be output. All the specifications that are also permitted in the BS2000 command `/SHOW-JV-ATTRIBUTES` are allowed. It is therefore also possible to make a partially qualified entry or use wildcards.
EDT does not perform any full syntax check.

If the specification does not designate any existing job variable then message `EDT4982` is output.

If the name of the job variable is fully qualified then the catalog ID (CATID) is only included in the output if it is already present in the name.

If `string` is not specified then all the job variables under the current user ID are output.

line — Line number as of which job variable information is to be written to the current work file.

If a line with a number greater than the previous highest line number is created then the current line number is modified.

If `line` is not specified then in the interactive mode's L mode, the result is output to `SYSOUT`, in batch mode it is output to `SYSLST` and in F mode it is written to work file `9`. Work file `9` is deleted before being used. If a file is open in work file `9` then the message `EDT5189` is output and the statement is not executed.

inc — Increment used to form the line numbers which follow `line`. If `inc` is not specified then the increment implicitly specified by `line` is used (see section “Implicit increment assignment” on page 35).

SHORT — Only the job variable names are output (default value).

LONG Further catalog information is output in addition to the job variable names.

Column	Header	Meaning
1-7	SIZE	Length of the current value in bytes
8	M	Indicates whether the job variable is a monitoring job variable (*/␣)
9-62	JOBVARIABLE NAME	Job variable name with USERID and possibly CATID (see above)
64-71	CR-DATE	Creation date (YY-MM-DD)
73	S	SHARE attribute (Y/N)
75	A	ACCESS attribute (W/R)
77	R	READ-PASS attribute (Y/N)
79	W	WRITE-PASS attribute (Y/N)

The list is alphabetically sorted on the job variable names.

If the output is sent to SYSOUT or SYSLST then it is accompanied by a header (see table). If the output is sent to work file 9 (in F mode, without the line operand) then the header is output in the information line (can be displayed using @PAR INFORMATION=ON). If the output is written to a work file due to the line operand then no header is output.

ISO4 The creation date (CR-DATE) is output in the form YYYY-MM-DD. The following fields (see table) are moved accordingly.

Any attempt to query the status of system job variables ($SYSJV.) is rejected with the message EDT3087.

If the *Job Variable Support* subsystem is not installed, the statement is rejected with the error message EDT5254. For details on job variables, see the JV User Guide [9].

Example

```
   1.00 90-06-27178<·························································
   2.00 GOOD MORNING, THE DATE TODAY IS 27.06.1990<·····························
   3.00 ·····································································



setjv 'today'=2·······································0001.00:00001(00)
```

The string present in line 2 is assigned to the job variable TODAY.

```
   1.00 90-06-27178<·························································
   2.00 GOOD MORNING, THE DATE TODAY IS 27.06.1990<·····························
   3.00 ·····································································



par global,information=on,index=off;stajv 'today' long·········0001.00:00001(00)
```

The information line is activated and the line number display is hidden. @STAJV is used to write information about the job variable TODAY to work file 9.

```
SIZE   M    JOBVARIABLE NAME                                    CR-DATE  S A R W
0000038 $EW.TODAY                                               90-06-27 N W N N
················································································



stajv'tod*'long4·······································0001.00:00001(09)
```

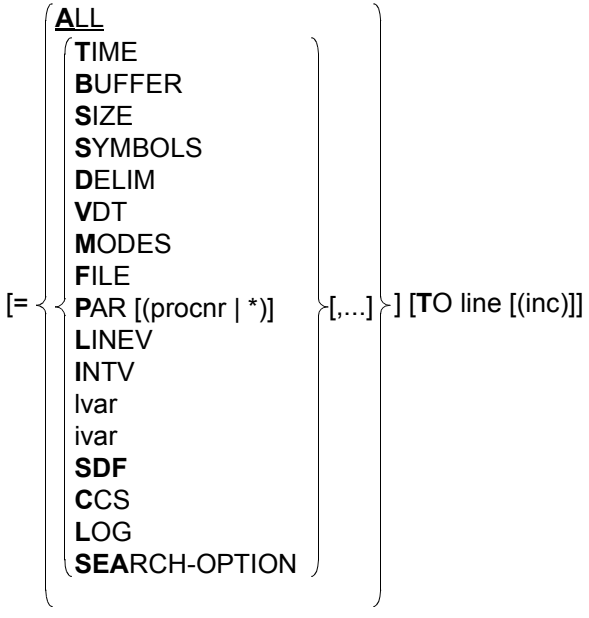
Information about the job variable is displayed.

```
SIZE   M    JOBVARIABLE NAME                                    CR-DATE    S A R
0000038 :XYZA:$EW.TODAY                                         1990-06-27 N W N
················································································



·······························································0001.00:00001(09)
```

Information about the job variable is displayed.

9.126 @STATUS – Display current settings and contents of variables

The @STATUS statement can be used to output the EDT and system environment settings together with the values of line number and integer variables.

Operation	Operands	F mode, L mode
@STATUS		

ALL — All the information concerning the parameters `TIME`, `BUFFER`, `SIZE`, `SYMBOLS`, `DELIM`, `VDT`, `MODES`, `LOG`, `SEARCH-OPTION`, `CCS` and `FILE` is output. In addition the user ID (USERID), the task serial number (TSN) and the current EDT operating mode are output (see section "Introduction to the EDT operating modes" on page 21).

TIME — This outputs the current time, the duration so far of the current EDT session, the CPU time required so far, the difference in the CPU time between the last two @STATUS statements.

BUFFER — This outputs the current size of the physical input/output buffer for the terminal (see `SETBF` macro) and the column number as of which the permissibility of the length of input in L mode is to be checked (see @CHECK statement). In batch mode, the column number is always output.

SIZE
: This operand is now only supported for reasons of compatibility. The value `0` is always output.

SYMBOLS
: The following are output: the current statement symbol (see @: statement), the valid delimiter for literals (see @QUOTE statement), the valid wildcards (see @SYMBOLS statement), the current range symbol together with the defined range (see @RANGE statement), the current fill character in hexadecimal form (see @SYMBOLS statement) and the current substitute character used to replace invalid characters during conversion operations (see the statement @PAR SUBSTITUTION-CHARACTER).

DELIM
: Outputs the set of declared text delimiter characters (see @DELIMIT statement).

VDT
: Outputs the number of screen lines and columns (see @VDT statement) together with the current behavior on screen output as defined using @PAR OPTIMIZE.

  In the case of 9763 terminals, the current screen format is also output (see @VDT statement).

MODES
: Outputs the default settings that may be defined using the @BLOCK, @CHECK, @INPUT, @TABS, @EDIT and @VTCSET statements.

  The settings for syntax control in L mode, the execution mode (see @SYNTAX statement) and the values that may be set for the @AUTOSAVE statement are also output.

FILE
: Outputs the global file name declared using the last @FILE statement. If a version number was also specified in the statement then this is output here.

  In addition (and if present), a local @FILE entry which has been defined implicitly (with @READ or @GET) or explicitly (with @FILE .. LOCAL) is output for every work file together with the name of an open file or the library and element name of an open library element.

  If the output from the `FILE` operand would be empty then it is omitted.

PAR (procnr | *)

A local @FILE entry (if present) together with the name of an open file or library element is first output for the specified work file in the same format as for @STATUS=FILE. Then the character set defined for this work file (see @CODENAME statement), the current values of all the work file-specific settings that can be modified using the @PAR statement which apply to the current work file and the work file-specific line number variables (*, ?, % and $) are output. Finally, the line and column positions (see @SETF statement) and the option selected for the line number display (see @PAR INDEX statement) are displayed for each (possible) screen window corresponding to the specified work file, together with the number of lines (after switching to F mode) in the visible area (see @PAR SPLIT) and an indication of whether (after switching to F mode), the cursor is located in this screen window.

If no work file is specified then a list containing the above-mentioned information is output for all the work files. If * is specified then the information is output for the current work file.

LINEV — Displays the contents of all the line number variables (#L0..#L20)

INTV — Displays the contents of all the integer variables (#I0..#I20)

lvar — The content of the named line number variable is output.

ivar — The content of the named integer variable is output.

SDF — The current SDF settings are displayed. In addition, for each work file, the name (if present) of the program set with @PAR SDF-PROGRAM or @SDFTEST is output together with an ID indicating whether this is an internal or an external name (as declared in @PAR SDF-NAME-TYPE). The format of the output is identical to the corresponding output for @STATUS=PAR.

CCS — The currently defined character set is output for the communications character set (TERMACT, see @CODENAME statement) as well as for every work file that has a character set and for every string variable. The character sets for the system files SYSDTA, SYSOUT and SYSLST and, in interactive mode, the character set defined using /MODIFY-TERMINAL-OPTIONS (TERMDEF) as well as the settings for the Auto mechanism are also output

LOG — The values defined for logging are output (see @LOG statement).

SEARCH-OPTION
: Outputs the default settings for the search functions (@ON statement) that have been defined using the @SEARCH-OPTION statement.

line
: Line number as of which information is to be written to the current work file.

  If a line with a number greater than the previous highest line number is created then the current line number is modified.

  If `line` is not specified then in the interactive mode's L mode, the result is output to `SYSOUT`, in batch mode it is output to `SYSLST` and in F mode it is written to work file 9. Work file 9 is deleted before being used. If a file is open in work file 9 then the message `EDT5189` is output and the statement is not executed.

inc
: Increment used to form the line numbers which follow `line`. If `inc` is not specified then the increment implicitly specified by `line` is used (see section “Implicit increment assignment” on page 35).

  If no operand is specified then the value `ALL` is used.

The output sequence when multiple operands are specified is predefined by EDT and not in any way dependent on the order in which the operands are entered. If the same operand is specified more than once then this does not cause the information concerning it to be output again.

Output to `SYSOUT` or `SYSLST` is sent in the character set that has been defined for these system files. If the output is written to a work file then it is sent in the work file's character set. If the work file has the character set `*NONE` then the character set `EDF041` is used. All characters which might be mapped to an invalid character when converted into the applicable output character set (for example, the values for @PAR SEPARATOR or @PAR STRUCTURE) are output both as characters (or possibly as blanks) and in the hexadecimal form `U'xxxx'`, in which case the character coding is displayed in `UTF16`.

If @SYNTAX TEST=ON has first been used to activate the test mode for L mode input and if @STATUS is entered in L mode then any specification of `TO line(inc)` is ignored, i.e. the output is written to `SYSOUT` instead.

9.127 @SUFFIX – Append strings

The @SUFFIX statement is used to append a string to every line or string variable in each specified range (see also @PREFIX).

Operation	Operands	F mode, L mode
@SUFFIX	{ lines / svars } [,...] **W**ITH string	

lines — One or more line ranges in which text is to be appended at the end of each line. Only existing lines are processed.

svars — One or more ranges of string variables in which text is to be appended at the end of each string variable.

string — String that is to be appended at the end of each line or string variable in each specified range. It is also permissible to specify an empty string.

The string is converted into the character set used by the work file or string variable. If the string contains characters which cannot be displayed in the target character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the @SUFFIX statement is rejected and error message EDT5453 is output.

If inserting the string would cause a line or string variable to exceed the maximum record length of 32768 characters then it is not inserted and the message EDT5474 is output.

If errors occur during processing (EDT5453 or EDT5474) then the statement is aborted. Any lines and/or string variables which have been successfully modified up to this point retain their changes.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Example

```
   1.00 AND<·····························································
   2.00 ONCE<····························································
   3.00 AGAIN<···························································
   4.00 AND<·····························································
   5.00 AND<·····························································
   6.00 ·····························································
suffix 4-5 with ' ONCE '·····································0001.00:00001(00)
```

The string ONCE is 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·····························································0001.00:00001(00)
```

The content of line 3 is 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·····················0001.00:00001(00)
```

First of all, 5 blanks are appended to lines 4 and 5 and then the content of line 4 is appended to these same 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 ·····························································
```

9.128 @SYMBOLS – Define symbols

The @SYMBOLS statement can be used to declare the wildcard symbols *asterisk* and *slash* for searches using placeholders (see section "Using wildcards in search terms" on page 80).

The FILLER operand declares a filler character (see section "Data window" on page 105).

Operation	Operands	F mode, L mode
@SYMBOLS	[{ **A**STERISK [= strspec1] / **S**LASH [= strspec2] / **F**ILLER [= strchar] } [,...]]	

ASTERISK= Declares the wildcard that stands for a string of any length including an empty string. When EDT starts, the value '*' is set by default. If the operand is not specified then the wildcard value is not modified unless no operand at all is specified, in which case the default value '*' is restored.

strspec1 Special character that is to be declared as the wildcard. If no value is specified, the default value '*' is restored.

SLASH= Defines the wildcard that stands for precisely one character. When EDT starts, the value '/' is set by default. If the operand is not specified then the wildcard value is not modified unless no operand at all is specified, in which case the default value '/' is restored.

strspec2 Special character that is to be defined as the wildcard. If no value is specified, the default value '/' is restored.

FILLER= Defines a filler character which is replaced by a blank when data is entered at the terminal in F mode. When EDT starts, the value X'00' is set by default. If the operand is not specified then the filler character is not modified unless no operand at all is specified, in which case the default value X'00' is restored.

strchar Any character that is to be declared as the filler character.

If the @SYMBOLS statement is entered without any operands then all 3 characters are reset to their default values.

Different characters must be specified for spec1 and spec2 as otherwise the statement is rejected with the message EDT3181. They must also be different from the characters defined in @QUOTE as otherwise the statement is rejected with the message EDT3180.

If spec1 or spec2 is not a valid special character then the @SYMBOLS statement is rejected with the error message EDT3952.

9.129 @SYNTAX – Set test mode

The @SYNTAX statement can be used to activate or deactivate test mode for input in L mode. If the test mode is activated then statements input in L mode are not executed (a syntax check of statements is always performed independently of the set test mode).

Operation	Operands	F mode, L mode
@SYNTAX	**TEST**MODE [= { **ON** / **OF** F }]	

TESTMODE= The TESTMODE operand defines whether or not the entered statements are to be executed. Test mode only applies to input in L mode.

ON The statements are subjected to a syntax check but are not executed (this also applies to external statement routines functioning as a statement filter). Exceptions to this rule are listed below. Data lines entered in L mode are not taken over into the work file. Input which starts with at least one EDT statement is checked for its syntax.

The following statements are always executed even if test mode is active:

- @HALT, @LOG, @RETURN, @SYNTAX and @: (redefine the EDT statement symbol)
- @STATUS. However, if test mode is active, the output is always sent to SYSOUT

The following statements and operands are not checked:

- Calls of external statement routines (statements with user statement symbol)
- The text operand in the statements @+, @-, @IF and @SET (format 6). If the statement otherwise contains no errors then the message EDT0110 is output instead of EDT0100 in the dialog.
- statements which contain indirect operand specifications.

OFF Test mode is deactivated.

Test mode is always deactivated when EDT is started.

The @STATUS statement can be used to output the current setting for the test mode.

In L mode, if test mode is active, then not only is the invalid statement identified. The position at which the error was detected is also marked with ： . If no errors are detected in a statement then the message `EDT0100` is output in the dialog.

Note

If the syntax of the `text` operand is to be checked in the statements @, @+, @- and @SET (format 6) then the statement must be split into two statements, e.g. @3:@... should be split into the two statements @3 and @...

9.130 @SYSTEM – Enter system commands

The @SYSTEM statement can be used to interrupt (like K2) the EDT session or to execute an operating system command without interrupting the EDT session.

Operation	Operands	F mode, L mode
@SYSTEM	[string [**T**O line [(inc)]]]	

string — String containing the command that is to be executed. The system expects a BS2000 command which is then executed immediately. The EDT session is then continued provided that the executed system command does not cause the program to be unloaded (see below). In this case, EDT is unloaded without regaining control again.

If `string` is not specified then the EDT session is interrupted. The `/RESUME-PROGRAM` or `/INFORM-PROGRAM` command can be used to continue the EDT session at the point where it was interrupted by @SYSTEM (see section "Interrupting an EDT session" on page 91). In uninterruptible system procedures, it is not possible to interrupt the EDT session in interactive mode.

line — Line number as of which any output from the system command is to be inserted in the current work file. It is only possible to insert output that the operating system writes to `SYSOUT` (this does not include, for example, formatted output from the `/SHOW-FILE` command). The command output is redirected, i.e. the output is written to the current work file and not to `SYSOUT`. When this is done, the output is converted from the character set `EDF041` into the character set used by the current work file.

If the output contains characters which are invalid in the current work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the error message `EDT5453` is issued.

If `line` is not specified then the output from the system command is written as specified in the command itself (usually to `SYSOUT`). If the command generates formatted output (e.g. `/SHOW-FILE`) then it may be necessary to refresh the work window with K3 (see section "Function keys in F mode" on page 123).

inc — Increment used to form the line numbers which follow `line`. If `inc` is not specified then the increment implicitly specified by `line` is used (see section "Implicit increment assignment" on page 35).

The @SYSTEM statement is one of the EDT statements with security implications (see also section "Access protection" on page 99). The statement is rejected in uninterruptible system procedures in interactive mode and on input from a file (read with RDATA from SYSDTA not equal to SYSCMD, execution of a start procedure).

The command can be written with or without the leading slash. Before being passed to the operating system, the command string is converted from the input source's character set into the character set UTFE.

It is only permissible to specify commands that can also be issued using the CMD macro. If a command is not permitted at the CMD interface or in the current SDF syntax file then it is rejected with the error message EDT4300 (which also contains the message key from the command return code). The commands permitted for use with the CMD macro are described in the Executive Macros User Guide [12].

Some system commands cause the program to be unloaded when used via the CMD interface (e.g. /EXIT-JOB, /LOGOFF, /HELP-SDF, /CALL-PROCEDURE, /START-PROGRAM, /LOAD-PROGRAM or user commands defined by means of SDF-A and implemented using command procedures). For a full overview of these commands, see the Executive Macros User Guide [12]. It is advisable to avoid unloading EDT due to this type of command since open files are not closed and there is no opportunity to write back unsaved work files.

Example

```
   1.00 ·····························································
   2.00 ·····························································
   3.00 ·····························································
   4.00 ·····························································
   5.00 ·····························································
   6.00 ·····························································




@SYSTEM '/SHOW-SYSTEM-INFORMATION' TO 1;@ON & F NOT 'OSD' DEL··0000.00:00010(00)
```

The output from the /SHOW-SYSTEM-INFORMATION command is to be used to obtain information about the OSD version.

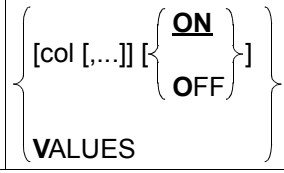
```
  12.00                  OSD-BC-VERSION  = V05.0C0000<·····························
  27.00 VM2000-MONITOR-  OSD-BC-VERSION  = *NONE<·····························
  25.00 ·····················································································
  26.00 ·····················································································
  27.00 ·····················································································
  28.00 ·····················································································
 ·····························································0012.00:00001(00)
```

The relevant information is stored in the work file.

9.131 @TABS (format 1) – Define and output hardware tabs

Format 1 of the @TABS statement is used to define tabulator (tab) positions for positioning with the hardware tabulator and output the current values of these positions. The hardware tabulator is only effective in F mode.

The settings for the hardware and software tabulator are stored separately. However, only one of the two tabulator functions can be active at any time. If the hardware tabulator is activated then any software tabulator that may be active is therefore deactivated.

Operation	Operands	F mode, L mode
@TABS		

col — Specifies one or more comma-separated tab positions for positioning with the hardware tabulator. The number of possible tab positions is restricted by the length of a statement and possibly also by the hardware. EDT goes to the tab positions in the specified order. If the tab positions are not specified in ascending order then the @TABS statement is not executed and the error message `EDT4940` is issued.

If the tab positions are defined in such a way that more fields per line would be required for the on-screen display than are permitted at the relevant terminal, then the @TABS statement is rejected with the error message `EDT5463` (at 975x terminals, for example, only 64 fields are permitted per line).

If `col` is not specified then the current values of the hardware tabulator's tab positions remain unchanged.

ON — The function of the hardware tabulator is activated (default value).

If the current or an earlier statement has defined tab positions then it is possible to specify TAR in order to move the cursor to the next defined tab position to the right of its current position. This corresponds to the *forward strategy* in the evaluation of software tabs.

If no tab positions are defined then the @TABS statement is rejected with the error message `EDT4941`.

OFF — The function of the hardware tabulator is deactivated. The defined tab positions are retained and can be reactivated using @TABS ON.

VALUES The current tab positions of the hardware tabulator are output. If no tab position is defined then there is no output.

The tab positions for output are five digits in length. Up to 11 tab positions can be output per output line. These are then followed by a line break.

In interactive mode, the output is written to SYSOUT and in batch mode it is written to SYSLST.

When EDT starts, the hardware tabulator is deactivated.

In EDIT-LONG mode, the hardware tabulator can only be used to go to the tab positions which have values smaller than the screen width. Tab positions with values greater than the screen width are ignored.

If the hardware tabulator is active and the @SCALE ON statement has been used in F mode to activate the column counter display then an additional screen line is displayed in which the current tab positions are indicated by an 'I'. This is not displayed in EDIT-LONG mode.

If the hardware tabulator is active then only EFG and AFG can be used to perform insertions and deletions within the tabulator positions since the hardware tabulator is implemented by means of terminal fields.

Example

```
  23.00 ·····························································
tabs 10,16,40·····················································0000.00:00001(00)
```

The values 10,16 and 40 are defined as tab positions for the hardware tabulator.

```
   1.00          BALR  14,15                  TO SUBROUTINE AND RETURN<·····
   2.00 LABEL    DC C  ' OK'<·····························································
   3.00 ·····························································
```

Entering TAB at the terminal moves the cursor to the tab positions 10,16 and 40.

9.132 @TABS (format 2) – Define and output software tabs

Format 2 of the @TABS statement is used to define tabulator (tab) characters and positions for positioning with the software tabulator, output the corresponding current values and activate and deactivate the software tabulator.

If software tabulators have been activated then tabulator expansion is performed when data lines are input from a terminal (not in batch mode or on input from procedures) in L mode (including indirect input via the text operand in certain statements) or when EDT enters/modifies data lines in the F mode data window. If the software tabulator has been defined (although not necessarily activated), then @TABS (format 3) can be used to perform tabulator expansion for existing lines. Tabulator expansion is performed from left to right in the data lines and it is possible to set two expansion strategies.

The normal *forward strategy* causes a tab character to be replaced by blanks (at least one) until the following character is located at the next tab position to the right of the current position.

The somewhat unusual *positioning strategy* causes the nth tab character to be positioned at the nth tab position irrespective of whether this is located to the left or the right of the current position. If it is located to the right of the current position then the tab character is replaced by the corresponding number of blanks in the same way as in the *forward strategy*. In contrast, if it is located to the left of the current position (or exactly at it) then the tab character is deleted and the current position is set to the tab position. Any following characters then usually overwrite any preceding characters in the entered line. The user can request a message to be output when this occurs.

In both strategies, any tab characters for which no further tab positions can be ascertained are not replaced and are retained as normal text characters.

The settings for the software and hardware tabulator are stored separately.
However, only one of the two tabulator functions can be active at any time. If the software tabulator is activated then any hardware tabulator that may be active is therefore deactivated.

Operation	Operands	F mode, L mode
@TABS	: : { [char [:] col [,...]] [{ **C**HECK / **F** ORWARD / **N**OCHECK } [col1]] / **ON** / **O**FF / **V**ALUES }	

char
: Character that EDT is to interpret as a tab character in subsequent input. This implicitly activates the software tabulator. Neither the statement symbol nor the separator character should be specified as the tab character. The character ; cannot be used when statements are entered in F mode since it is interpreted as a statement separator. The tab character `char` must be separated from `col` by a : if `char` is one of the characters `C`, `F`, `O`, `V`, `N` or a digit.

  If `char` is not specified but a strategy is then only the strategy is modified.

  If neither `char` nor a strategy is specified then the current settings for the software tabulator are set to undefined and the software tabulator is deactivated.

col
: Specifies the comma-separated tab positions for positioning with the tab character. The number of possible tab positions is only restricted by the length of a statement. If the tab positions are not specified in ascending order then the @TABS statement is not executed and the error message `EDT4940` is issued.

CHECK
: EDT uses the *positioning strategy* described in the introduction and outputs a message if the position is set to a tab position to the left of the current position in the line. If this occurs when text is output at the terminal then the message `EDT2902` is output. If it occurs during the evaluation of tab characters in a work file using the @TABS statement (format 3) then the message `EDT4312` is output. Furthermore, in this case, no further tabulator expansion is performed as of the line to which the message refers.

  If `CHECK` but not `char` is specified then the strategy is set irrespective of whether a software tabulator is defined and active. It applies until it is explicitly modified again.

FORWARD
: EDT uses the *forward strategy* described in the introduction.

  If `FORWARD` but not `char` is specified then the strategy is set irrespective of whether a software tabulator is defined and active. It applies until it is explicitly modified again.

NOCHECK
: EDT uses the *positioning strategy* described in the introduction and does not check whether any tabulator expansion is performed at a position to the left of the current line position.

  If `NOCHECK` but not `char` is specified then the strategy is set irrespective of whether a software tabulator is defined and active.

  It applies until it is explicitly modified again.

  When EDT starts, the value `NOCHECK` is set by default.

col1 Specifies the number of characters per line (1..32768) for the check of the line length in L mode. EDT checks the number of characters in newly entered lines. In particular, it is used to display any cases in which the predefined line length is exceeded due to possible tabular expansions.

If a line is longer than the value specified in col1 then the line is nevertheless created and EDT outputs the message EDT2901 to indicate that the predefined number of characters per line has been exceeded.

If col1 is not specified then the set value remains unchanged.

When EDT starts, the value for the line length check in L mode is set to 32768 characters.

ON The function of the software tabulator is activated. A software tabulator must already have been defined.

If no tab positions are defined then the @TABS statement is rejected with the error message EDT4941.

OFF The function of the software tabulator is deactivated. The defined tab positions are retained and can be reactivated using the @TABS ::ON statement.

VALUES The current tab character and the associated software tabulator tab positions are output. If no tab character is defined then there is no output.

The tab positions for output are five digits in length. Up to 11 tab positions can be output per output line. These are then followed by a line break. The tab character is not output in the continuation lines.

In interactive mode, the output is written to SYSOUT and in batch mode it is written to SYSLST.

If, due to tabulator expansion, a line exceeds the maximum length of 32768 characters then it is truncated to the maximum length and the message EDT1903 is output in L mode and the message EDT2400 in F mode.

If tab positions are defined and the @SCALE ON statement has been used in F mode to activate the column counter display then an additional screen line is displayed in which the current tab positions are indicated by an 'I'. In this line, the tabulator character itself is depicted in the statement code column.

The tab character applies globally to all the work files. If data is entered in F mode then tabulator expansion is performed in the entire line, not just in the displayed section.

If only a strategy is specified (e.g. @TABS::FORWARD), then this strategy is set even if no software tabulator has been defined. If *no* strategy is specified then the *set* strategy is retained. When EDT starts, the value NOCHECK is set by default.

Note

If @TABS:: is entered then the current software tabulator setting is set to undefined. If no software tabulator at all is specified (but, for example, a hardware tabulator is) then this statement has no effect.

It is better to use @CHECK to modify the value for the line length check in L mode. Here it is now only supported for reasons of compatibility.

The main use of the @TABS statement is to create files, e.g. source files, which have the correct information in the correct columns. For example, it makes sense to issue the statement @TABS::[:10,16,40 CHECK 71 when creating a source file for an Assembler program. `CHECK` ensures that a message is output if excessively long names are specified (`FORWARD` instead of `CHECK` would not achieve the desired effect here). It is sensible to specify a maximum line length of 71 since this means that column 72, which indicates continuation lines, cannot be overwritten.

Caution

It is risky to have too many tab characters in a line when using the *positioning strategy*. If a further tabulator expansion is performed, e.g. due to a correction in the line in F mode then these all suddenly become effective – usually with unwanted consequences.

Example

```
  23.00 ·····························································
lower off;tabs ::[: 10,16,40·····································0000.00:00001(00)
```

`[` is declared as the tab character. Let the tab positions be 10, 16 and 40.

```
   1.00 [balr[14,15[subroutine and then return<·····························
   2.00 label[dc[c' ok'<····················································
   3.00 ·····························································
```

The text is entered together with 3 tab characters.

```
   1.00          BALR  14,15                  SUBROUTINE AND THEN RETURN<·····
   2.00 LABEL    DC    C  ' OK'<·····················································
   3.00 ·····························································
```

Tabulator expansions are performed in columns 10, 16 and 40.

9.133 @TABS (format 3) – Expand software tabs in work files

Format 3 of the @TABS statement is used to expand software tabs in work files and string variables if a tab character and corresponding tab positions have been defined (see @TABS, format 2).

Operation	Operands	F mode, L mode
@TABS	**R**ANGE [=⎧lines⎫ [,...]] ⎩svars⎭	

lines One or more line ranges in which tab characters are to be expanded.

svars One or more ranges of string variables in which tab characters are to be expanded.

If neither `lines` nor `svars` is specified then the tab characters are expanded in all the lines of the current work file.

If no software tabulator is defined then the statement is rejected with the error message `EDT5953`. However, the software tabulator does not have to be active.

If, due to tabulator expansion, a line or string variable exceeds the maximum length of 32768 characters then tabulator expansion is aborted and the message `EDT4314` is output. There is no check of the maximum length defined using @TABS (format 2) or @CHECK (format 1).

If the *positioning strategy* for the software tabulator is set with a check (CHECK in format 2) and *backward positioning* would be performed then tabulator expansion is aborted and the message `EDT4312` is output.

9.134 @TMODE – Output task attributes

The @TMODE statement provides the user with information about the task under which EDT is running. The information is output as a message.

Operation	Operands	F mode, L mode
@TMODE		

The following information about the task under which EDT is running is output.

TSN	Task sequence number
USERID	User ID specified in the `LOGON` command
ACCOUNT	Account number of the task
CPU-TIME	Used CPU time
DATE	Date (`YYYY-MM-DD`)
TIME	Time
STATEMENT SYMBOL	The current statement symbol (e.g. `@`)
TERMINAL	Type of terminal

Example

```
  23.00 ·····························································
tmode·····························································0000.00:00001(00)
```

Information about the task attributes is requested.

```
  22.00 ·····························································
%  EDT0300 0582 BOND     007         15.1635 2005-09-27 15:12:19 @ 9763
···························································0000.00:00001(00)
```

9.135 @UNLOAD – Unload a module

The @UNLOAD statement is used to unload load units that were loaded with @USE or @RUN. At the same time, application routines that were assigned to the unloaded unit are deactivated and their user statement symbols are set to invalid.

Operation	Operands	F mode, L mode
@UNLOAD	{ **U**NIT=entry **M**ODULE=entry (name) }	

UNIT=entry The load unit (UNIT) with the name entry is to be completely unloaded. When loading external statement routines with @USE or user routines with @RUN, EDT informs the dynamic binder loader of the name of the specified module or the entry point (ENTRY) in the form of a UNIT name. When this name is specified in @UNLOAD then the complete load unit including any dynamically autolinked modules is unloaded.

MODULE=entry
The module with the name entry is to be unloaded. In contrast to the specification of UNIT, only the specified module is unloaded.

name Name of the module that is to be unloaded. This format is now only supported for reasons of compatibility.

If the @UNLOAD statement results in a load unit or module being unloaded, EDT then makes the necessary changes in the list of declared statement routines. Here, a statement routine is only identified by means of the string specified in entry or name without taking account of the load structure. All statement routines with the specified name are deactivated and their user statement symbols are set to invalid. Statement routines defined using @USE ...,ENTRY=* are ignored during this operation, i.e. the corresponding user statement symbol remains valid and the unloaded load unit is loaded again, if necessary, with the corresponding entry point the next time it is called.

The @UNLOAD statement is one of the EDT statements with security implications (see also section “Access protection” on page 99). The statement is rejected in uninterruptible system procedures in interactive mode and on input from a file (read with RDATA from SYSDTA not equal to SYSCMD, execution of a start procedure).

The attempt to unload modules that have been dynamically loaded internally by EDT is rejected with the message EDT1907.

The same message is output if the module cannot be unloaded for other reasons. The possible causes for this include an incorrect module name, the specification of a module that has been loaded as shareable or the specification of a `CSECT` or `ENTRY` name that did not result in a load operation in the @USE or @RUN statement.

Note

Names of up to 32 characters in length are permitted in the `entry` specification and the name is case-sensitive. Only 8 characters are permitted for `name` and any lowercase characters that are entered are converted into uppercase.

Caution

On @UNLOAD, EDT only deletes the user statement symbol to which the specified load unit or specified module were assigned directly.
User statement symbols which refer to other entry points (`ENTRY`) in the load unit are retained and subsequently point to invalid addresses. It is the user's own responsibility to ensure that such user statement symbols are no longer used after the unload.

9.136 @UNSAVE – Delete SAM or ISAM file

The @UNSAVE statement deletes a SAM or ISAM file and the corresponding catalog entry.

Operation	Operands	F mode, L mode
@UNSAVE	file [(ver)]	

file — Name of the file that is to be deleted. The name must correspond to the SDF data type <filename 1..54>.

Here, the symbolic name '/' for a file for which the LINK name EDTSAM or EDTISAM has been assigned by means of the SET-FILE-LINK command is not permitted.

ver — Version number of the file that is to be deleted. If the specified version number does not match the file's version number, the statement is rejected with message EDT4985.

If the specified file does not exist or cannot be accessed as required then the statement is rejected with a corresponding error message.

9.137 @USE – Define external statement routines

The @USE statement is used to define user statements by specifying a user statement symbol and an associated statement routine (see Subroutine Interfaces User Guide [1]).

Operation	Operands	F mode, L mode
@USE	**C**OMMAND='[spec]' [,**E**NTRY={entry / *} [,**M**ODLIB=modlib] / ({name / *} [,modlib])]	

spec — Special character which is used as the user statement symbol for an external statement routine. The user statement symbol must be different from the current EDT statement symbol.

In the specific case that an external statement routine is to be used as a statement filter then an empty string must be specified instead of a user statement symbol. However, this is only possible when EDT is called as a subroutine (see the section “Special application as a statement filter” in [1]).

entry — Entry point for the external statement routine: The module which contains the entry point is loaded immediately.

name — Entry point for the external statement routine: The operand syntax with parentheses is now only supported for reasons of compatibility.

* — The name of the statement routine's entry point is specified as the statement name when the user-defined statement is entered. The module which contains the entry point is not loaded until the user-defined statement is executed for the first time.
Similarly, the `UNIT` name which is used to unload the load unit again (see below) is not formed until this point.

modlib — Name of the library containing the module which contains the entry point. The library must exist. Otherwise, the statement is rejected with the error message `EDT5372`.

If the module containing the entry point is not found in the specified library, the system first searches in the alternative libraries `BLSLIBnn` then in the private task library and system task library `$TASKLIB`.

If no library is specified, the system first searches in the private task library and then in the system task library `$TASKLIB`.

If the search fails, the error message `EDT5372` is issued.

A maximum of 5 different user statement symbols can be declared. The attempt to declare a sixth user statement symbol is rejected with the message `EDT5373`.

If no other operands are specified apart from the user statement symbol then the statement routine previously defined using the user statement symbol is deactivated.

If the execution of the @USE statement results in a separate load operation then a `UNIT` name which is the same as the entry point specified in `entry` or `name` is notified to the dynamic binder loader.

This `UNIT` name can be specified in the @UNLOAD statement in order to unload all the load units that were loaded together with the entry point. If the entry point is found inside another load unit, @USE does not therefore result in a separate load operation and the entry point can only be unloaded together with this load unit.

If the value `*` is specified instead of `entry` or `name` then this mechanism is not applied until a user-defined statement is entered. This means that the names of all the entry points which have resulted in a load operation are notified to the dynamic binder loader as `UNIT` names.

EDT only dynamically loads an entry point if this has not already been found in a previously loaded load unit. If EDT itself has been loaded as a subroutine in a user program, this also applies to entry points in this user program.

It is not therefore possible to use entry points of the same name in different load units in parallel. It is nevertheless possible that the dynamic binder loader may identify duplicate names when loading a statement routine. In this case, the @USE statement or user statement that caused dynamic loading is rejected with the message `EDT4208`. If EDT is not running in F mode then the error message from the dynamic binder loader is also output. This message makes it possible to identify the name of the duplicate symbol.

This situation can be avoided by using the @UNLOAD statement to unload the load unit loaded with a previous @USE statement before issuing any further @USE statements. The associated user statement symbol is then also canceled.

In contrast, if the user statement symbol is simply canceled using @USE COMMAND='spec' or if a user statement symbol is assigned twice then the originally assigned load unit is not implicitly unloaded.

If EDT-specific entry points or module names are specified in the @USE statement then it is rejected with the message `EDT4933`.

If the ENTRY=* or (*,modlib) operand is specified then conversion to uppercase characters is always performed when the name of the entry point is formed from the first part of the user statement. The remainder of the user statement may or may not be converted to uppercase depending on the @PAR LOW setting.
If (*,modlib) is specified then (for reasons of compatibility) only a maximum of 8 characters are taken into account during the formation of the entry point name. Otherwise, a maximum of 32 characters are taken into account.

For details on the implementation of external statement routines or statement filters and for information on passing parameters to these, see the section “Calling a user-defined statement” in [1].

The @USE statement is one of the EDT statements with security implications (see also section “Access protection” on page 99). Under certain privileged IDs, the @USE statement is rejected. This also applies to uninterruptible system procedures in interactive mode (read from SYSDTA with RDATA, execute EDT start procedure) unless the @USE statement is issued by the protected procedure itself (SYSDTA=SYSCMD).

Note

Names of up to 32 characters in length are permitted in the entry specification and the name is case-sensitive. Only 8 characters are permitted for name and any lowercase characters that are entered are converted into uppercase.

Example 1

The module which contains the entry point JOBVAR in the library PRIVLIB can be called with the parameters CATJV <name>, ERAJV <name>, GETJV <name>,<ln> or SETJV <name>,<ln>. The name of the entry point JOBVAR is specified directly in the @USE statement.

```
@USE COMMAND = '*',ENTRY=JOBVAR,MODLIB=PRIVLIB ---------------------------(1)
*CATJV JV.TEST ------------------------------------------------------------ (2)
*SETJV JV.TEST,3 ---------------------------------------------------------- (3)
```

(1) The user statement symbol * is declared and the module JOBVAR is loaded.

(2) The module JOBVAR is called with the parameter 'CATJV JV.TEST'.

(3) The module JOBVAR is called with the parameter 'SETJV JV.TEST,3'.

Example 2

The library PRIVLIB contains the two modules SORT and HELP. The name of the entry point is not defined until the user-defined statement is actually called.

```
@USE COMMAND = '*',ENTRY=*,MODLIB=MODLIB ------------------------------ (1)
*SORT 20-100 ---------------------------------------------------------- (2)
*HELP EDT5100 --------------------------------------------------------- (3)
```

(1) * is declared as the user statement symbol. However, no module is loaded as yet.

(2) The module SORT is loaded and called with the parameter '20-100'.

(3) The module HELP is loaded and called with the parameter 'EDT5100'.

9.138 @VDT – Control screen format

The @VDT statement can be used to set the screen format in F mode.

Operation	Operands	F mode, L mode
@VDT	[{**F1** / **F2** / **F3** / **F4**}]	

F1 Sets the screen format to 24 lines and 80 columns. This format is also set when EDT starts or if no operand is specified.

F2 Sets the screen format to 27 lines and 132 columns.

F3 Sets the screen format to 32 lines and 80 columns.

F4 Sets the screen format to 43 lines and 80 columns.

The screen formats `F2`, `F3` and `F4` are only supported by DSS 9763. If a terminal does not support a specified format then the statement is rejected with the error message `EDT4945`.

If the @VDT statement is entered in a statement sequence or in a statement block in `BLOCK` mode then it is the last statement to be executed after all the others have been processed.

The @VDT statement implicitly terminates the display of two work windows.

In batch mode, this statement is ignored and no error message is issued. In L mode and when output is sent to `SYSOUT` (provided that `SYSOUT` is assigned to a terminal) then the screen format `F1` is always set.

9.139 @VTCSET – Control screen output

When output is sent to SYSOUT (e.g. by means of one of the statements @PRINT or @ON..PRINT), the @VTCSET statement specifies whether the line mode control characters which may be present in the lines of data that are to be output are transferred unchanged or are converted into smudge characters.

Operation	Operands	F mode, L mode
@VTCSET	[{**ON** / **OF** F}]	

ON Specifies that the output is not checked and that therefore no smudge characters are used as replacements. This is also the setting when EDT is started.

OFF Causes line mode control characters in data lines to be replaced by smudge characters when output to SYSOUT.

The character specified in /MODIFY-TERMINAL-OPTIONS SUBSTITUTE-CHARACTER=... is used as the smudge character.

The setting @VTCSET OFF is of use when the data which is interpreted as control characters fragments the screen output. However, this is only possible when output is written to SYSOUT and SYSOUT is assigned to the terminal. The @VTCSET statement has no effect in the case of formatted work window output in F mode, output to SYSLST (@LIST) or output in batch mode.

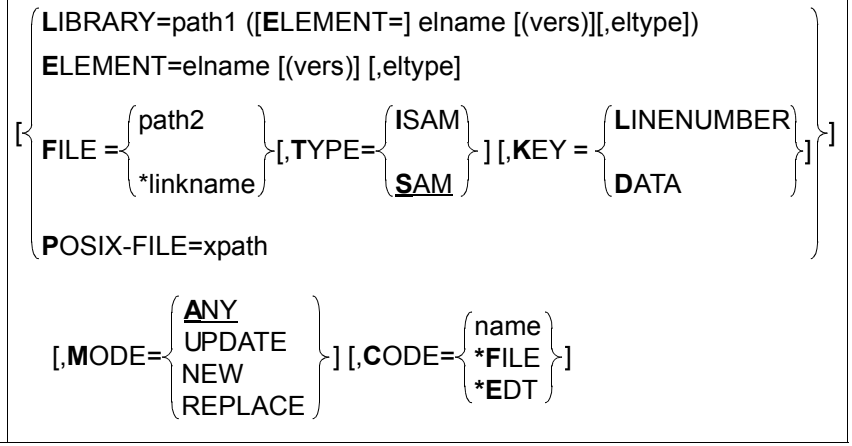
Output to SYSOUT is usually written in the character set defined for SYSOUT (either that of the terminal or that of the file or library element to which SYSOUT is assigned). The output may therefore be converted.

Once this conversion has been performed, the line mode control characters are replaced (see Executive Macros User Guide [12], VTCSET macro).
Since all the terminals supported by EDT only support character sets in which the control characters have the same binary coding, the sequence in which the characters are replaced is only of significance for the depiction of the smudge character.

9.140 @WRITE (format 1) – Write file

The @WRITE statement (format 1) creates a new file and writes the content of the current work file to the new file, overwrites an existing file with the content of the current work file or writes the content of the current work file back to a file opened using @OPEN (format 1). An open file remains open when @WRITE is issued. If existing files are overwritten then the old file content is completely replaced. The work file is retained in all cases.

Whenever this section refers to a “file”, this can be a SAM file, an ISAM file, a library element or a POSIX file.

Operation	Operands	F mode, L mode
@WRITE		

LIBRARY=... A library element is to be overwritten. This is defined by explicitly specifying the library name and the element designation.

path1 Name of the library.

elname Name of the element.

vers Version of the required element (see the LMS User Guide [14]). If `vers` is not specified or if `*STD` is specified then the highest available version of the element is selected.

eltype Type of element. Permitted type specifications are `S`,`M`, `P`, `J`, `D`, `X`, `*STD` as well as freely selectable type names having one of these types as basic type. If `eltype` is not specified then the default type specified with @PAR ELEMENT-TYPE is used. The permitted element types and their meanings are described in chapter “File processing” on page 131.

ELEMENT=... A library element is to be written. This is defined by means of the element designation without any library name specification. The default library set with @PAR LIBRARY is used implicitly (if @PAR LIBRARY has been specified, otherwise the error message EDT5181 is issued).

The operands elname, vers and eltype have the same meaning as when a library is specified explicitly (see above).

FILE= A BS2000 file is to be written.

path2 Name of the BS2000 file (fully qualified file name) that is to be written.

*linkname File link name of the BS2000 file that is to be written. The file name and the file attributes are stored in the Task File Table. In this way, it is possible to create files with nonstandard names. The file link name must not be specified as the special file name *BY-PROGRAM. This results in the error EDT4923. If no file link name is defined then the statement is rejected with the message EDT5480.

If the file link name is declared as the special file name *DUMMY then it is treated as an non-existent file. However, no file is created.

TYPE= Specifies the access method when creating a new file. In the case of existing files, this operand is ignored.

SAM A SAM file is created and written. This is the default value.

ISAM An ISAM file is created and written.

KEY= In the case of ISAM files, specifies how the ISAM key is to be formed. In the case of other file types, this operand is ignored.

If the operand is not specified then the ISAM key is formed from the line number when a new file is written or when an existing file is overwritten. When data is written back to an open file, the ISAM key is formed from the line number if KEY=LINENUMBER or KEY=IGNORE was specified when the file was opened. If KEY=DATA was specified when the file was opened then the ISAM key is taken over from the data area.

LINENUMBER

The ISAM key is formed from the line number. If the position of the key differs from the default value or if the key is too long, the message EDT5465 is output and the file is not written. If the key is too short, the line number is truncated from the left.

DATA The ISAM key is a component of the data range in the work file. In this case, the user must make sure that the sequence of work file records corresponds to the sequence of ISAM keys as otherwise the write operation will be rejected with the message EDT4208 (DMS error code 0AAB).

POSIX-FILE= A POSIX file is to be written.

xpath Path name of the POSIX file that is to be written.

The `xpath` operand can also be specified as a string variable. It must be specified as a string variable if the path name contains characters which have a special meaning in EDT syntax (e.g. blanks, semicolons in F mode or commas).

MODE= Specifies whether the file should already be present. If an open file is to be written back, this operand is ignored.

ANY If the file already exists then it is overwritten. Otherwise, it is created and written. This is the default value.

UPDATE The file that is to written must already be present as otherwise the message `EDT5281`, `EDT5270` or `EDT5310` is output depending on the file type. The old content is completely overwritten.

NEW The file is created and written. It must not already be present as otherwise the message `EDT5258`, `EDT5273` or `EDT5311` is output depending on the file type.

REPLACE Has the same meaning as `ANY`. If the file already exists then it is overwritten. Otherwise, it is created and written.

CODE= The operand controls the character set in which the work file is to be written.

If this operand is not specified then the character set defined with @PAR CODE is used for POSIX files and the work file's character set is used for other files. If in the case of SAM files, ISAM files or library elements, the character set of an existing file differs from that of the work file then the message `EDT5457` is output in batch mode and no write operation is performed. In interactive mode, the following query is output:

```
% EDT0915 CONVERT TO FILE CCS (&00)? REPLY (Y=YES; N=NO)?
```

If the user responds `Y` then a conversion to the file's character set is performed before the write operation. If the user responds `N` then the work file's character set is used.

name Character set that is to be used for writing. The name of a valid character set must be specified for `name` (see section “Character sets” on page 47).

*FILE Before the write operation, the work file is converted into the character set of the existing SAM file, ISAM file or library element or into the character set used when opening a POSIX file. If this character set was *NONE then EDF03IRV is used. If the file does not yet exist or if an existing POSIX file is to be overwritten then the message EDT1181 is output and the CODE operand is ignored. System behavior is then the same as when the CODE operand is omitted.

*EDT The work file's character set is used for writing irrespective of whether any file that may exist has a different character set.

When new files are written or when existing files are overwritten, it is always necessary to specify a file name operand. When files opened with @OPEN are written back, the file name operand can be omitted. If the file name operand is omitted even though no file is open then the statement is rejected with the message EDT5122. If all the statement's operands are omitted and no file is open then the statement is interpreted as @WRITE (format 2) and a @FILE entry is searched for (see @WRITE format 2).

If the specified file cannot be accessed as required then the statement is rejected with a corresponding error message.

After the write operation, the employed character set is entered in the catalog for SAM files, ISAM files and library elements. If this character set is EDF03IRV and the file that is to be written already exists with the character set *NONE in the catalog then this value is retained.

If the work file is converted before writing and if it contains characters which are invalid in the character set used by the file that is to be written then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the file is not written and error message EDT5453 is output. The user can then define a substitute character or modify the character set for writing and run @WRITE again.

If the work file contains lines that are too long for the file that is to be written (e.g. if the file has a fixed record length) or if the conversion operation creates any such records (possible in the case of Unicode character sets), then the write operation is aborted with the message EDT5444.

If, during the processing of an opened ISAM file, the character set is changed either from or to UTF16 or if this occurs implicitly due to a corresponding specification in the CODE operand a file opened with @OPEN cannot be written back since this would modify the length of the key field. In this case, the @WRITE statement is rejected with the error message EDT5468.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Caution

Since the default setting is MODE=ANY, existing files are overwritten without any warning being issued.

Example

```
@WRITE LIBRARY=PROGLIB(ELEMENT=SYNT)
```

The current work file is written to the element SYNT in the library PROGLIB. In this case, the highest possible version and the default type specified with @PAR ELEMENT-TYPE are used.

```
@PAR LIBRARY=LIB1
@SET #SO2='PROC.PR'
@WRITE ELEMENT=.#SO2 (V01),J
```

The current work file is written to the element with the name PROC.PR, the version V01 and the element type J in the library LIB1.

```
@WRITE FILE=FILE2,MODE=NEW,CODE=*EDT
```

The SAM file FILE2 is created and the current work file is written to the new file. The work file's character set is used.

```
@OPEN POSIX-FILE=/home/user1/test/data,CODE=UTF8
@WRITE ,MODE=ANY
```

The current work file is written back to the open POSIX file data in the directory /home/user1/test.

9.141 @WRITE (format 2) – Write SAM file

The @WRITE statement (format 2) fully or partially writes the content of the current work file to disk or tape as a SAM file.

Operation	Operands	F mode, L mode
@WRITE	[file] [(ver)] [,] [lines[,...]] [:cols[,...]:] [**K**EY] [{**U**PDATE | **O**VERWRITE}]	

file — Name of the SAM file that is to be written. The name must correspond to the SDF data type `<filename 1..54>` or must consist of the special specification `'/'`.

If the file `file` does not yet exist then it is created prior to the write operation. If the `file` operand is not specified then the explicit local @FILE entry is used as the file name if present. If not, the global @FILE entry (from the @FILE statement) is used and, failing this, the implicit local @FILE entry (e.g. from the @READ statement). If there is no @FILE entry then @WRITE is rejected with the message `EDT5484`.

If the specified file cannot be accessed as required then the statement is rejected with a corresponding error message.

If the file link name `EDTSAM` is assigned to a file then the user simply needs to specify `'/'` in order to write this file (see chapter "File processing" on page 131).

ver — Version number of the file that is to be overwritten. If an incorrect version number is specified for an existing file then the statement is rejected with `EDT4985`. If the file does not yet exist then this specification is ignored and version `001` of the file is written.

lines — One or more line ranges that are to be written to the SAM file. If lines are specified more than once then they are also written more than once.

If `lines` is not specified, then the entire file is written.

cols — One or more column ranges which define the section to be written for each record. The ranges may repeat and overlap. The column specifications refer to the *characters* in the current work file. If column values which exceed the work file record length are specified then blanks are written to the file in their place.

If no column range is specified then the lines are written in full.

KEY When the SAM file is written, every line is prefixed with an 8-character long key which is derived from the associated line number. As a result, the file can subsequently be read in using exactly the same line numbers (see @READ with the KEY operand).

UPDATE Specifying UPDATE causes the lines that are to be saved to be appended at the end of the existing SAM file.

This operand is ignored if no SAM file with the specified name exists.

OVERWRITE An existing file of the same name is overwritten without any request for confirmation. If the specified file does not yet exist, OVERWRITE has no effect.

If neither UPDATE nor OVERWRITE is specified and if a file with the same name already exists then, in interactive mode, EDT issues the query:

```
%  EDT0903 FILE 'file' IS IN THE CATALOG, FCBTYPE = fcbtyp
%  EDT0296 OVERWRITE FILE? REPLY (Y=YES; N=NO)?
```

If the user answers the message with Y then the existing file is overwritten as a SAM file with the content of the current work file. In contrast, if the user answers N then the file is not written and the message EDT0293 is output.
In batch mode, the file is always overwritten.

If an existing file is overwritten by @WRITE without the UPDATE operand then the file type and/or file attributes may change. The file is written as a SAM file with default attributes (e.g. variable record length) unless a corresponding /SET-FILE-LINK command with the file link name EDTSAM has already been specified with the divergent attributes (see section “File processing” on page 131). Files of the type PAM or BTAM cannot be overwritten.

The file is only opened temporarily during the write operation.

The character set used for the write operation depends on whether the file is overwritten, created or extended (see the UPDATE operand.

If the file is overwritten or created then the data is written in the work file's character set and this character set is entered for the file in the catalog.

If the file is extended then the data is converted from the work file's character set into the character set specified in the file's catalog entry. If the value *NONE is entered for the file in the catalog then EDF03IRV is used (see also section “Character sets” on page 47).

If the work file contains characters which are invalid in the character set of the file that is to be written then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the file is not written and the error message EDT5453 is output. This does not apply to invalid characters outside of the line or column range that is to be written. These characters are ignored.

If the work file contains lines that are too long for the file that is to be written (e.g. if the file has a fixed record length) or if the conversion operation creates any such records (possible in the case of Unicode character sets), then the write operation is aborted with the message EDT5444.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

If all the statement's operands are omitted and no file is open in the current work file then the statement is interpreted as @WRITE (format 1) and no @FILE entry is searched for (see @WRITE format 1).

Caution

If the @FILE statement has been used to prefix a file name then the statement sequence

```
@READ 'filename'
@WRITE
```

does not cause the read file to be written. Instead, the file from the @FILE entry is written.

Example

```
1.     A VERY SHORT FILE ------------------------------------------- (1)
2.     @WRITE 'TEST.@WRITE.1' -------------------------------------- (2)
2.     @FILE 'TEST.@WRITE.1' --------------------------------------- (3)
2.     @WRITE UPDATE ----------------------------------------------- (4)
2.     @DELETE ----------------------------------------------------- (5)
1.     @READ ------------------------------------------------------- (6)
3.     @PRINT
1.0000 A VERY SHORT FILE
2.0000 A VERY SHORT FILE ------------------------------------------- (7)
3.
```

(1) A line is written to the work file.

(2) This line is written to disk as the file TEST.@WRITE.1.

(3) The file name TEST.@WRITE.1 is declared via @FILE.

(4) @WRITE refers to the file name declared in (3). UPDATE causes the content of the work file – i.e. still the line created in (1) – to be appended at the end of the file TEST.@WRITE.1.

(5) The content of the work file is deleted.

(6) The file TEST.@WRITE.1 is moved into the work file (here again, there is no need to specify a file name.)

(7) It can be seen that the line was appended at the end of the file in (4).

9.142 @XCOPY – Read POSIX file

The @XCOPY statement is used to read a POSIX file which is stored in the POSIX file system into the current work file. This statement is now only supported for reasons of compatibility. In its place, users are recommended to use @COPY (format 1) with the operand POSIX-FILE.

Operation	Operands	F mode, L mode
@XCOPY	**F**ILE=xpath [,**C**ODE= { name / **E**BCDIC / **I**SO }]	

xpath — Path name of the POSIX file that is to be read into the current work file.

The xpath operand can also be specified as a string variable. It *must* be specified as a string variable if the path name contains characters which have a special meaning in EDT syntax (e.g. blanks, semicolons in F mode or commas).

If the specified file does not exist or cannot be accessed as required then the statement is rejected with a corresponding error message.

CODE= — Defines the character set that is to be assumed for the POSIX file. Since it is not possible to assign character sets to POSIX files in the POSIX file system, a user specification is required here.

If CODE is not specified then the character set defined in @PAR CODE is assumed. When EDT starts, the value EDF041 is set.

- name — Character set of the POSIX file that is to be read in. The name of a valid character set must be specified for name (see section “Character sets” on page 47).
- EBCDIC — The keyword EBCDIC is now only supported for reasons of compatibility and is a synonym for the character set EDF041.
- ISO — The keyword ISO is now only supported for reasons of compatibility and is a synonym for the character set ISO88591.

The records read from the file are inserted after the last line in the current work file using the procedure “Insertion between two lines” (see section “Line number assignment” on page 36).

If the current work file is empty and has the character set *NONE then it is assigned the character set specified in CODE. If no character set is specified then the work file is assigned the character set defined using @PAR CODE.

If the work file already has a character set then the records that are to be read in are converted from the file's character set into the work file's character set.

If the file that is to be read in contains characters which cannot be displayed in the work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the file is not read in and the error message EDT5453 is output.

If the file is present in a Unicode character set and contains an illegal byte sequence, e.g. surrogate characters, then it will be impossible to read it even if SUBSTITUTION-CHARACTERS is specified. In this case, the read operation is rejected with the message EDT5454.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

9.143 @XOPEN – Open and read a POSIX file

The @XOPEN statement is used to open a POSIX file which is stored in the POSIX file system and read it into the current work file. This statement is now only supported for reasons of compatibility. In its place, users are recommended to use the @OPEN (format 1) statement with the operand `POSIX-FILE`.

Operation	Operands	F mode, L mode
@XOPEN	**F**ILE=xpath [,**C**ODE= { name / **E**BCDIC / **I**SO }] [,**M**ODE= { **A**NY / UPDATE / NEW / REPLACE }]	

xpath — Path name of the POSIX file that is to be opened.

The `xpath` operand can also be specified as a string variable. It *must* be specified as a string variable if the path name contains characters which have a special meaning in EDT syntax (e.g. blanks, semicolons in F mode or commas).

If the specified file does not exist or cannot be accessed as required then the statement is rejected with a corresponding error message.

CODE= — Defines the character set that is to be assumed for the POSIX file. Since it is not possible to assign character sets to POSIX files in the POSIX file system, a user specification is required here.

If `CODE` is not specified then the character set defined in @PAR CODE is assumed. When EDT starts, the value `EDF041` is set.

- name — Character set of the POSIX file that is to be opened. The name of a valid character set must be specified for `name` (see section “Character sets” on page 47).
- EBCDIC — The keyword `EBCDIC` is now only supported for reasons of compatibility and is a synonym for the character set `EDF041`.
- ISO — The keyword `ISO` is now only supported for reasons of compatibility and is a synonym for the character set `ISO88591`.

MODE= Specifies whether the file should or may already be present.

- ANY: If the file already exists then it is opened for processing and read in. Otherwise, it is created and opened for processing. This is the default value.
- UPDATE: The file that is to be opened for processing and read in must already exist as otherwise the message EDT5310 is output.
- NEW: The file is created and opened for processing. It must not already be present. Otherwise, the message EDT5311 is output.
- REPLACE: If the file already exists then it is opened for processing. However, its previous content is deleted and is not read into the work file. If the file does not exist, it is created and opened for processing.

If the current work file is not empty then the @XOPEN statement is rejected with the message EDT5191. Unlike BS2000 files or library elements, POSIX files can be opened multiple times in different work files (in POSIX, files are not protected against being simultaneously opened by different tasks).
The records read from the file are inserted in the current work file after position 0.0000 using the procedure “Insertion between two lines” (see section “Line number assignment” on page 36).
If the empty work file has the character set *NONE then it is assigned the character set specified in CODE. If no character set is specified then the work file is assigned the character set defined using @PAR CODE.

If the empty work file already has a character set (e.g. due to a preceding @CODENAME) then the records that are to be read in are converted from the file's character set into the work file's character set. If the file that is to be read in contains characters which cannot be displayed in the work file's character set then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the file is not read in and the error message EDT5453 is output.

If the file is present in a Unicode character set and contains an illegal byte sequence, e.g. surrogate characters, then it will be impossible to read it even if SUBSTITUTION-CHARACTERS is specified. In this case, the read operation is rejected with the message EDT5454.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Note

When EDT is terminated (@HALT, @END, @RETURN), if a file is open due to @XOPEN and the save query EDT0900 is output, then the POSIX file name is displayed in the form X=xpath.

9.144 @XWRITE – Save content of current work file in a POSIX file

The @XWRITE statement can be used to write the content of the current work file to a POSIX file. The work file is retained. This statement is now only supported for reasons of compatibility. In its place, users are recommended to use the @WRITE (format 1) statement with the operand `POSIX-FILE`.

Operation	Operands	F mode, L mode
@XWRITE	[**F**ILE=xpath] [,**C**ODE={name / **E**BCDIC / **I**SO}] [,**M**ODE={**A**NY / UPDATE / NEW / REPLACE}]	

xpath — Path name of the POSIX file that is to be written.

The `xpath` operand can also be specified as a string variable. It must be specified as a string variable if the path name contains characters which have a special meaning in EDT syntax (e.g. blanks, semicolons in F mode or commas).

If `xpath` is not specified then a POSIX file opened with @XOPEN or @OPEN (format 1) is written back. If there is no open POSIX file then the statement is rejected with the message `EDT5122`.

If the specified file cannot be accessed as required then the statement is rejected with a corresponding error message.

CODE= — Defines the character set with which the POSIX file is to be written.

If `CODE` is not specified then the file is written in the character set defined in @PAR CODE (including when writing back files that were opened with another character set). When EDT starts, the value `EDF041` is set.

- name — Character set of the POSIX file that is to be written. The name of a valid character set must be specified for `name` (see section “Character sets” on page 47).
- EBCDIC — The keyword `EBCDIC` is now only supported for reasons of compatibility and is a synonym for the character set `EDF041`.
- ISO — The keyword `ISO` is now only supported for reasons of compatibility and is a synonym for the character set `ISO88591`.

MODE= Specifies whether the file should or may already be present. If an open file is to be written back, this operand is ignored.

- ANY If the file already exists then it is overwritten. Otherwise, it is created and written. This is the default value.
- UPDATE The file that is to be written must already exist as otherwise the message EDT5310 is output.
- NEW The file is created and written. It must not already be present. Otherwise, the message EDT5311 is output.
- REPLACE Has the same meaning as ANY. If the file already exists then it is overwritten. Otherwise, it is created and written.

If a POSIX file has been opened with @XOPEN or @OPEN (format 1) then it is not necessary to specify the file name in @XWRITE. The content of the opened file is replaced by the content of the work file. The file remains open until @CLOSE is issued.

If the work file is converted before writing and if it contains characters which are invalid in the character set used by the file that is to be written then these characters are replaced by a substitute character provided that such a character has been specified (see @PAR SUBSTITUTION-CHARACTER); otherwise, the file is not written and error message EDT5453 is output. The user can then define a substitute character or modify the character set for writing and run @XWRITE again.

If the statement is interrupted with K2 and the EDT session is continued with /INFORM-PROGRAM then the processing of the statement is aborted and message EDT5501 is output.

Caution

Since the default setting is MODE=ANY, existing files are overwritten without any warning being issued if MODE is not specified.

9.145 0..22 – Switch work file

This statement causes EDT to switch to another work file.

Operation	Operands	F mode
0..22		

EDT displays the work file selected with the `0..22` statement in the work window in which the statement was entered.

The line position and column position are set to the values that were previously valid in the newly set work file. If the work file has not yet been used then the default values apply.

The @SETF statement can also be used to change work file and, at the same time, set the position to any required line and column.

10 Statement codes in F mode (alphabetical)

This chapter contains a detailed description of all the EDT statement codes available in F mode listed in alphabetical order.

In the case of statements which contain special characters, *alphabetical* means the sequence defined in the character set EBCDIC.DF.04.

10.1 + – Move forward in the work window

The statement code + repositions the work window so that the line in which + was entered becomes the first line in the data window.

Statement code	Key
+	DUE or F2

This statement code does not modify the column position.

10.2 + – Move forward in work window by structure depth

If the statement code + is sent with the function key F1 then the position is set to the next record that has the same structure depth as the specified record.

The structure depth is the distance of the first non-blank character from the start of the record. It makes no difference whether the record is displayed as of column 1 or whether the screen section has been shifted to the right.

If a structure symbol other than a blank has been defined (see @PAR STRUCTURE) then only records which contain this character are considered. If the blank is defined as the structure symbol then all the records are considered. The default value for the structure symbol is @.

Statement code	Key
+	F1

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

If the specified record does not contain a structure symbol then the statement code is rejected with the message EDT5354.

Example

This example assumes that the blank has been defined as the structure symbol using @PAR STRUCTURE=' '.

```
   1.00 if (b != 0)<·····························································
   2.00 {<·····································································
   3.00     if (a == 1)<·····························································
+  4.00     {<·····································································
   5.00          b = 0;<·····························································
   6.00          c = 3;<·····························································
   7.00     }<·····························································
   8.00     else<·····································································
   9.00     {<·····································································
```

```
·····························································0001.00:00001(00)
```

The statement code + is entered in line 4.00 and is sent with F1.

```
   7.00     }<·····························································
   8.00     else<··························································
   9.00     {<·····························································
 
 
 
 
·····························································0007.00:00001(00)
```

The position has been moved to the next record with the same structure depth.

10.3 * – Delete copy buffer

The statement code * deletes a copy buffer created with C, M or R.

Statement code	Key
*	DUE or F2

The statement code * is evaluated before A, B, O, C, M or R irrespective of the line in which * is entered. This means that the copy buffer is always deleted if * is entered in a line.

The deletion of the copy buffer is acknowledged with the message EDT0292. The message is not issued if new lines have been entered in the copy buffer by simultaneously specifying the statement codes C, M or R.

10.4 – – Move backward in work window

The statement code – repositions the work window so that the line in which – was entered becomes the last line in the data window.

Statement code	Key
–	DUE or F2

This statement code does not modify the column position.

The statement code – has no effect if there are not enough records to completely fill the work window.

10.5 – – Move backward in work window by structure depth

If the statement code – is sent with the function key F1 then the position is set to the previous record that has the same structure depth as the specified record.

The structure depth is the distance of the first non-blank character from the start of the record. It makes no difference whether the record is displayed as of column 1 or whether the screen section has been shifted to the right.

If a structure symbol other than a blank has been defined (see @PAR STRUCTURE) then only records which contain this character are considered. If the blank is defined as the structure symbol then all the records are considered. The default value for the structure symbol is @.

Statement code	Key
–	F1

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

If the specified record does not contain a structure symbol then the statement code is rejected with the message EDT5354.

Example

This example assumes that the blank has been defined as the structure symbol using @PAR STRUCTURE=' '.

```
   1.00 if (b != 0)<·························································
   2.00 {<···································································
   3.00     if (a == 1)<·····················································
   4.00     {································································
   5.00          b = 0;<·····················································
   6.00          c = 3;<·····················································
-  7.00     }<·······························································
   8.00     else<····························································
   9.00     {<·······························································



·······························································0001.00:00001(00)
```

The statement code – is entered in line `7.00` and is sent with [F1].

```
   4.00     {<·······························································
   5.00          b = 0;<·····················································
   6.00          c = 3;<·····················································
   7.00     }<·······························································
   8.00     else<····························································
   9.00     {<·······························································



·······························································0004.00:00001(00)
```

The position has been moved to the previous record with the same structure depth.

10.6 A – Copy or move after a line

The statement code A copies or moves records with line numbers collected in the copy buffer using C, M or R *after* the specified line. If the copy buffer has been filled with the statement codes C or M then it is subsequently deleted.

Statement code	Key
A	DUE or F2

For the sake of simplicity; the description below refers only to *copying* even if the lines are deleted after the copy operation (i.e. they are moved).

If the copy buffer is empty then the statement is rejected with the message EDT5376.

When inserting or appending the copied lines, EDT assigns line numbers using the procedure “Insertion between two lines” (see section “Line number assignment” on page 36). If it is not possible to insert the lines then the copy operation is not performed, the copy buffer is not deleted and the message EDT5365 is output.

If the current work file is empty and has the character set *NONE then it is assigned the character set of the source work file of the first line for copying when the copy operation is performed.

If the current work file has a character set then the lines that are to be copied are converted into this work file's character set.

If characters are detected which cannot be displayed in the work file's character set then these are replaced by a substitute character if such a character has been specified (see @PAR SUBSTITUTION-CHARACTER). Otherwise, the statement code is rejected and the error message EDT5453 is output.

Note

The statement code A is not executed until the statements C, M and R have been processed. As a result, the target location can be specified in a work window before the lines that are to be copied in a single dialog step.

Example

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
c  2.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
a  3.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································************************
```

Line 2.00 is to be copied to a position after line 3.00.

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
   2.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   3.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   3.10 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································************************
```

Line `2.00` has been copied as the new line `3.10` after line `3.00`.

10.7 B – Copy or move before a line

The statement code B copies or moves records with line numbers collected in the copy buffer using C, M or R *before* the specified line. If the copy buffer has been filled with the statement codes C or M then it is subsequently deleted.

Statement code	Key
B	DUE or F2

For the sake of simplicity; the description below refers only to *copying* even if the lines are deleted after the copy operation (i.e. they are moved).

If the copy buffer is empty then the statement is rejected with the message EDT5376.

When inserting or appending the copied lines, EDT assigns line numbers using the procedure “Insertion between two lines” (see section “Line number assignment” on page 36). If it is not possible to insert the lines then the copy operation is not performed, the copy buffer is not deleted and the message EDT5365 is output.

If statement code B has been entered in the first screen line then the screen is subsequently repositioned in such a way that the inserted lines are visible.

If the current work file is empty and has the character set *NONE then it is assigned the character set of the source work file of the first line for copying when the copy operation is performed.

If the current work file has a character set then the lines that are to be copied are converted into this work file's character set. If characters are detected which cannot be displayed in the work file's character set then these are replaced by a substitute character if such a character has been specified (see @PAR SUBSTITUTION-CHARACTER). Otherwise, the statement code is rejected and the error message EDT5453 is output.

Note

The statement code B is not executed until the statements C, M and R have been processed. As a result, the target location can be specified in a work window before the lines that are to be copied in a single dialog step.

10.8 C – Collect lines for copying

The statement code C transfers the line numbers and work file number corresponding to the specified record to the EDT copy buffer so that the record can subsequently be copied using one of the statement codes A, B or O.

Statement code	Key
C	DUE or F2

As soon as one of the statement codes A, B or O is entered, the copy operation is executed, i.e. the records are inserted in the corresponding position. The content of the copy buffer is then deleted.

If, when C is specified, the copy buffer already contains entries that were generated using the statement codes M or R then the copy buffer is deleted before the specified line is entered and the message EDT0295 is output.

The copy buffer can also be filled with lines taken from different work files. The records determined by the copy buffer can then be copied into any work file.

The copy buffer contains the work file and line numbers of the records collected with C. The line numbers and content of the records that are to be copied should therefore not be modified between being collected with C and the performance of the copy operation with the statement codes A, B or O. However, if this does occur then the new contents are copied. If any records have been deleted in the interim then these are skipped without any warning during the copy operation.

Since the copy buffer only contains work file and line numbers, no conversion of the selected lines is performed during collection. Since the target character set is not determined until the target work file is specified, any conversion that is required is not performed until the statement codes A, B or O are evaluated (see section “Character sets” on page 47).

Note

If the screen is split then lines can be copied from the first to the second work window in a single dialog step. If items are copied from the second to the first work window then two dialog steps may be necessary depending on the processing sequence.

Example

```
   1.00 EDT is the BS2000  file<·····································
   2.00 editor, used for the user-<·····································
c  3.00 <·····························································
   4.00 friendly creation and editing<·····································
   5.00 of BS2000 files in SAM and ISAM formats<···························
   6.00 as well as text-like library<·····································
b  7.00 elements and POSIX files.<·····································
a  8.00 The repetitive operations which occur<·····························
   9.00 during editing, such as deleting<·····································
  10.00 ·····························································
```

Line `3.00` is to be copied to a position before line `7.00` and after line `8.00`. This is achieved by entering statement code `C` in line `3.00`, statement code `B` in line `7.00` and statement code `A` in line `8.00`.

```
   1.00 EDT is the BS2000  file<·····································
   2.00 editor, used for the user-<·····································
   3.00 <·····························································
   4.00 friendly creation and editing<·····································
   5.00 of BS2000 files in SAM and ISAM formats<···························
   6.00 as well as text-like library<·····································
   6.10 <·····························································
   7.00 elements and POSIX files.<·····································
   8.00 The repetitive operations which occur<·····························
   9.00 during editing, such as deleting<·····································
  10.00 ·····························································



  21.00 ·····························································
%  EDT5360 NO COPY. BUFFER EMPTY
·····························································0001.00:00001(00)
```

Line `3.00` has been copied to a position before line `7.00` but not after line `8.00`. Instead, EDT issues an error message.

Lines collected with `C` can only be copied to a single target location since the copy buffer is deleted after the first copy operation. The second specified target location causes the error message. If lines are to be copied multiple times then they must be collected with `R`.

10.9 D – Delete records

The statement code D deletes the specified record from the work file.

Statement code	Key
D	DUE or F2

Example

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
d  2.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   3.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································************************
```

The line 2.00 is to be deleted. This is achieved by specifying D in the statement code column.

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
   3.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································************************
```

Line 2.00 has been deleted.

10.10 D – Delete record mark

If the statement code D is sent with the function key F3 then it deletes any record mark that was present (see section “Record marks” on page 45).

Statement code	Key
D	F3

The statement code D only deletes the record marks 1..9; special marks are retained.

10.11 E – Insert characters

Statement code E sets the specified line to overwritable for the subsequent insertion of characters. If necessary, space is created for the insertion of the characters.

Statement code	Key
E	DUE or F2

If the line identified by E does not contain at least 20 NULL characters at the line-end then EDT provides 20 NULL characters at the end of the line. The characters in the line, (including the closing LZE) that are shifted by the 20 NULL characters remain present in the data window but are no longer visible.

The user can insert up to 20 characters anywhere in the line (EFG). If fewer than 20 characters are inserted then the shifted remainder of the line is moved back into the data window after data transfer.

In EDIT-LONG mode, the statement code E causes a line with NULL characters to be made available in the data window in addition to the usual record display.

Example

```
   1.00 EDT is the BS2000  file<·····································
   2.00 editor, used for the user-<·····································
   3.00 <·····························································
   4.00 friendly creation and editing<·····································
   5.00 of BS2000 files in SAM and ISAM formats<·····························
e  6.00 as well as text-like library elements and POSIX files.<·····················
   7.00 The repetitive operations which occur<·····························
   8.00 during editing, such as deleting<·····································
  10.00 ·····························································
```

Statement code E is entered in line 6.00 in order to insert characters.

```
   1.00 EDT is the BS2000  file<·····································
   2.00 editor, used for the user-<·····································
   3.00 <·····························································
   4.00 friendly creation and editing<·····································
   5.00 of BS2000 files in SAM and ISAM formats<·····························
   6.00 as well as text-like library elements and POSIX fil·····················
   7.00 The repetitive operations which occur<·····························
   8.00 during editing, such as deleting<·····································
  10.00 ·····························································
```

Since there are fewer than 20 characters available at the end of line 6.00, EDT shifts the remainder of the line (including the LZE) out of the data window and provides 20 NULL characters.

```
  1.00 EDT is the BS2000 file<·····························································
  2.00 editor, used for the user-<·····························································
  3.00 <·····························································
  4.00 friendly creation and editing<·····························································
  5.00 of BS2000 files in SAM and ISAM formats<·····························
  6.00 as well as text-like and library elements and POSIX fil·····
  7.00 The repetitive operations which occur<·····························
  8.00 during editing, such as deleting<·····························
 10.00 ·····························
```

The word *and* has been inserted in line 6.00.

```
  1.00 EDT is the BS2000  file<·····························
  2.00 editor, used for the user-<·····························
  3.00 <·····························
  4.00 friendly creation and editing<·····························
  5.00 of BS2000 files in SAM and ISAM formats<·····························
  6.00 as well as text-like and library elements and POSIX files.<···········
  7.00 The repetitive operations which occur<·····························
  8.00 during editing, such as deleting<·····························
 10.00 ·····························
```

Since fewer than 20 characters were inserted in line 6.00, EDT moves the shifted line remainder back into the data window.

10.12 H – Activate hexadecimal mode for a record

The statement code H activates hexadecimal mode for the selected record only (see section “Hexadecimal mode” on page 120) and sets all the screen lines that belong to this record to overwritable.

Statement code	Key
H	DUE or F2

If the screen window (when the screen is split) is too small to display the data line with all its hex lines then the statement code is rejected with the message EDT2404.

If necessary, and possible, the data window is repositioned in a way that permits the display of the hex lines.

The hex lines are not displayed when the screen is reconstructed after being sent unless the statement code H is entered in the statement code column again. However, for this to be possible the data window and the statement code column must first have been set to overwritable with @PAR EDIT-FULL=ON.

Example

The data in the example is coded in UTF8.

```
   1.00 Price change:<·····················································
   2.00 <··································································
H  3.00       200,00 €<·····················································
```

Only line 3.00 is to be displayed in hexadecimal mode. This is achieved by specifying H in the statement code column.

```
   1.00 Price change:<·····················································
   2.00 <··································································
   3.00       200,00 €<·····················································
        2222223332332E·····················································
        000000200C0002·····················································
        ·············8·····················································
        ·············2·····················································
        ·············A·····················································
        ·············C·····················································
        ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
```

10.13 I – Activate permanent insert function

The statement code I activates the permanent insert function, i.e. new lines are provided on the screen *before* the specified line. They can be filled with text and inserted in the work file after data transfer. This process repeats until the permanent insert function is terminated (see below); i.e. after data transfer, another range of new lines is made available after the lines that have just been inserted.

For the distinction between new lines and empty lines (which correspond to records of length 0) see section “F mode” on page 101.

Statement code	Key
I	DUE or F2

The permanent insert function is terminated if

- there is no input in the last new line output or
- the statement code S is entered or
- other statements move the insertion position out of the data window.

If the work window is large enough then an insertion range of 9 new lines is provided. The area displayed in the data window may be moved to make the new lines available.

The permanent insert function is only activated if at least 10 screen lines are visible in the data window (not including the column counter). If the data window is shorter than this then I simply provides one insertion range of the length of the data window – 1.

When inserting the new lines, EDT assigns line numbers using the procedure “Insertion between two lines” (see section “Line number assignment” on page 36). If insertion is not possible then the statement code I is rejected with message EDT5365.

If no text is entered in one of the available new lines then no record is created in the work file. However, this does not terminate the permanent insert function provided that the line in question is not the last line.

Example

```
   1.00 a<·····························································
   2.00  b<····························································
   3.00    c<··························································
   4.00     d<·························································
   5.00      e<························································
   6.00 111<···························································
   6.10 112<···························································
   6.20 113<···························································
   6.30 114<···························································
   7.00    222<························································
i  8.00       333<·····························································
   9.00 ·····························································
```

More than 9 lines are to be inserted in front of line `8.00`. This is achieved by entering the statement code `I` (permanent insert function) in line `8.00`.

```
   1.00 a<·····························································
   2.00  b<····························································
   3.00    c<··························································
   4.00     d<·························································
   5.00      e<························································
   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 ·····························································
·····························································0001.00:00001(00)
```

All 9 lines in the inserted range have been 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 the insertion range has been filled, an additional insertion range consisting of 9 new lines with line number increment `0.01` is made available.

10.14 J – Join two records

The statement code J appends the specified record to the preceding record. The appended record is then deleted.

Statement code	Key
J	DUE or F2

If the total of the record lengths of the records joined in this way exceeds the maximum permitted record length of 32768 characters then it is truncated to the maximum record length. In this case, the record (partially) appended with J is not deleted and the message EDT2400 is output.

If the statement code J is issued for the first record in the work file then it is ignored.

The opposite operation – separating a record – is described in section “Statement in data window – splitting a record” on page 112 (see also the description of the @SEP statement).

Example

```
   1.00 EDT is the BS2000  file<·····------------------------------------
   2.00 editor, used for the user-<·····------------------------------------
   3.00 <·····------------------------------------------------------------
j  4.00 friendly creation and editing<·····------------------------------------
   5.00 of BS2000 files in SAM and ISAM formats<·····---------------------------
   6.00 as well as text-like library<·····------------------------------------
j  7.00 elements and POSIX files.<·····------------------------------------
   8.00 The repetitive operations which occur<·····---------------------------
   9.00 during editing, such as deleting<·····--------------------------------
  10.00 ·····------------------------------------------------------------
```

The records in lines 4.00 and 7.00 are each to be joined to the record that directly precedes them.

```
   1.00 EDT is the BS2000  file<·····------------------------------------
   2.00 editor, used for the user-<·····------------------------------------
   3.00 friendly creation and editing<·····------------------------------------
   5.00 of BS2000 files in SAM and ISAM formats<·····---------------------------
   6.00 as well as text-like and library elements and POSIX files.<···········
   8.00 The repetitive operations which occur<·····---------------------------
   9.00 during editing, such as deleting<·····--------------------------------
  10.00 ·····------------------------------------------------------------
```

The records have been appended. Appending a record to a record of length 0 is equivalent to moving the appended record.

10.15 K – Copy a line to the statement line

The statement code K copies the specified screen line to the statement line.
The copy operation starts at the specified column position. Starting at this point, it is possible to copy at most the number of characters that will fit in the statement line. The last character in the statement line is set to binary zero. Any previous content in the statement line is first deleted.

Statement code	Key
K	DUE or F2

At least one character in the line copied with K must be overwritten, modified or added in the statement line if it is to be sent as a statement.

The statement line is interpreted in the character set that has been defined for communication with the terminal (see statement @CODENAME name,TERMINAL). It may therefore be necessary to convert the line from the work file's character set to this target character set. If, in such a case, it is not possible to convert individual characters then the message EDT5453 is issued. The statement is output nevertheless and the non-converted characters are replaced by question marks '?'.

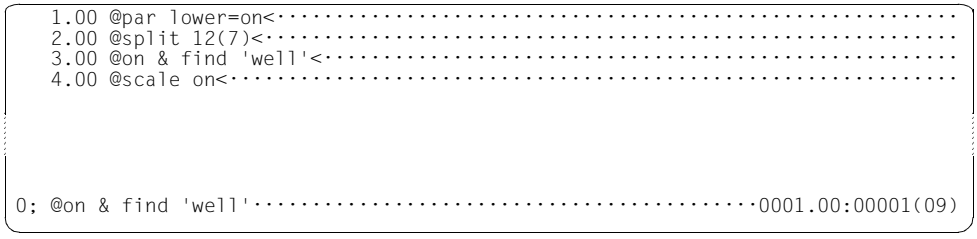
Example

```
   1.00 EDT is the BS2000 file<·····························································
   2.00 editor, used for the user-<·····························································
@SHIH·····························································0001.00:00001(00)
```

The @SHIH statement outputs the EDT statement buffer to work file 9.

```
   1.00 @par lower=on<·····························································
   2.00 @split 12(7)<·····························································
k  3.00 @on & find 'well'<·····························································
   4.00 @scale on<·····························································
·····························································0001.00:00001(09)
```

Statement code K copies screen line 3.00 into the statement line.



0 is inserted in front of the copied statement. This switches to work file 0 and starts the search.

10.16 L – Convert lines into lowercase

The statement code L causes the specified record to be converted into lowercase. Conversion is performed in the same way as for @CONVERT TO=LOWER using the corresponding XHCS function.

Statement code	Key
L	DUE or F2

Example

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
   3.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
L  5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
```

Line 5.00 is to be converted into lowercase characters.

```
   1.00 BERGER   ADALBERT  HOCHWEG 10     81234 MUENCHEN<·····················
   3.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   4.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   5.00 stiwi    manuela   postweg 3      80123 muenchen<·····················
```

Line 5.00 has been converted into lowercase characters.

10.17 M – Collect lines for move

The statement code M transfers the line numbers and work file numbers corresponding to the specified record to the EDT copy buffer so that the record can subsequently be moved using one of the statement codes A, B or O.

Statement code	Key
M	DUE or F2

As soon as one of the statement codes A, B or O is entered, the move operation is executed, i.e. the records are inserted in the corresponding position and are deleted at their original position. The content of the copy buffer is then deleted.

If, when M is specified, the copy buffer already contains entries that were generated using the statement codes C or R then the copy buffer is deleted before the specified line is entered and the message EDT0295 is output.

The copy buffer can also be filled with lines taken from different work files. The records determined by the copy buffer can then be moved into any work file.

The copy buffer contains the work file and line numbers of the records collected with M. The line numbers and content of the records that are to be copied should therefore not be modified between being collected with M and the performance of the copy operation with the statement codes A, B or O. However, if this does occur then the new contents are moved. If any records have been deleted in the interim then these are skipped without any warning during the move operation.

Since the copy buffer only contains work file and line numbers, no conversion of the selected lines is performed during collection. Since the target character set is not determined until the target work file is specified, any conversion that is required is not performed until the statement codes A, B or O are evaluated (see section “Character sets” on page 47).

Note

If the screen is split then lines can be moved from the first to the second work window in a single dialog step. If items are moved from the second to the first work window then two dialog steps may be necessary depending on the processing sequence.

Example

```
   1.00 EDT is the BS2000 file<·····························································
   2.00 editor, used for the <·····························································
m  3.00 <·····························································································
   4.00 friendly creation and editing<·····································
a  5.00 of BS2000 files in SAM and ISAM formats<·····························
   6.00 as well as text-like library<·····································
   7.00 elements and POSIX files.<·····································
   8.00 The repetitive operations which occur<·····························
   9.00 during editing, such as deleting<·····································
  10.00 ·····························································
```

Line `3.00` is to be moved to a position after line `5.00`. This is achieved by entering statement code `M` in line `3.00`, statement code `A` in line `5.00`.

```
   1.00 EDT is the BS2000  file<·····························································
   2.00 editor, used for the user-<··········
   4.00 friendly creation and editing<·····
   5.00 of BS2000 files in SAM and ISAM formats<·····
   5.10 <·····
   6.00 as well as text-like library<·····
   7.00 elements and POSIX files.<·····
   8.00 The repetitive operations which occur<·····
   9.00 during editing, such as deleting<·····
  10.00 ·····
```

Line `3.00` has been moved to a position after line `5.00`.

10.18 O – Copy or move on a line range

The statement code O copies or moves records with work file and line numbers collected in the copy buffer using C, M or R *onto* the specified line and the subsequent line range.

Statement code	Key
O	DUE or F2

For the sake of simplicity; the description below refers only to copying even if the lines are deleted after the copy operation (i.e. they are moved).

If the copy buffer is empty then the statement is rejected with the message EDT5376.

It is necessary to distinguish between two cases:

1. If the specified record is displayed as of column 1 in the data window then the entire content of the specified record is overwritten by the record that is to be copied.
2. If the specified record is displayed in the data window as of a column position greater than column 1 then the range starting at the displayed column position is overwritten with the content and to the length of the record that is to be copied. If the length of the record that is to be copied is less than the length of the record specified in O then the remaining parts of the record that is to be overwritten are retained. In this way it is possible to insert records in other records or append them to other records.

If more than one line is transferred for copying then they are written to the corresponding number of records which follow the specified record. If this operation goes beyond the end of the work file then new lines are created there.

When inserting or appending the copied lines, EDT assigns line numbers using the procedure “Insertion between two lines” (see section “Line number assignment” on page 36). If it is not possible to append the lines then the copy operation is not performed, the copy buffer is not deleted and the message EDT5365 is output. However, any lines that have already been overwritten remain so.

If the current work file is empty and has the character set *NONE then it is assigned the character set of the source work file of the first line for copying when the copy operation is performed.

If the current work file has a character set then the lines that are to be copied are converted into this work file's character set. If characters are detected which cannot be displayed in the work file's character set then these are replaced by a substitute character if such a character has been specified (see @PAR SUBSTITUTION-CHARACTER). Otherwise, the statement code is rejected and the error message EDT5453 is output.

Note

The statement code O is not executed until after any C-, M and R statements have been processed. As a result, it is possible to specify the target location before the lines that are to be copied in a single dialog step in a work window.

Unexpected effects may occur if O is used to copy over a line which is itself located in the range of lines that are to be moved or copied, and in particular when lines are moved with M. For more information, see the following examples:

Example 1

```
   1.00 1<·····························································
c  2.00  2<·····························································
o  3.00   3<·····························································
```

Line 2.00 is to be copied to line 3.00. This is achieved by entering statement code C in line 2.00, statement code O in line 3.00.

```
   1.00 1<·····························································
   2.00  2<·····························································
   3.00  2<·····························································
```

Line 3.00 has been overwritten with the content of line 2.00.

Example 2

```
   1.00 1<·····························································
c  2.00  2<·····························································
c  3.00   3<·····························································
c  4.00    4<·····························································
c  5.00     5<·····························································
   6.00      6<·····························································
```

Lines `2.00` to `5.00` are collected for copying.

```
   1.00 1<·····························································
   2.00  2<·····························································
   3.00   3<·····························································
o  4.00    4<·····························································
   5.00     5<·····························································
   6.00      6<·····························································
```

The lines from the copy buffer are to be copied onto line `4.00` and the following lines.

```
   1.00 1<·····························································
   2.00  2<·····························································
   3.00   3<·····························································
   4.00  2<·····························································
   5.00   3<·····························································
   6.00  2<·····························································
   7.00   3<·····························································
```

Since the copy operation has modified the content of the lines in the copy buffer itself, the above result is obtained.

Example 3

```
   1.00 1<·····································································
m  2.00  2<·····································································
   3.00   3<·····································································
```

Line `2.00` is selected to be moved.

```
   1.00 1<·····································································
o  2.00  2<·····································································
   3.00   3<·····································································
```

It is to overwrite line `2.00`.

```
   1.00 1<·····································································
   3.00   3<·····································································
```

Line `2.00` has been deleted because, when lines are moved, they are deleted after transfer.

Example 4

```
c  1.00 -111-222<·····························································
c  2.00 -333-444<·····························································
c  3.00 -555-666<·····························································
   4.00 ·····································································



>8·····························································0001.00:00001(00)
```

Lines `1.00` to `3.00` are selected for copying. At the same time, the statement `>8` is used to move the data window 8 characters to the right.

```
o  1.00 <·····································································
   2.00 <·····································································
   3.00 <·····································································
   4.00 <·····································································









<<·····························································0001.00:00009(00)
```

The statement code `0` copies the lines from the copy buffer over line 1 which has been moved to the right and the line which follows it. At the same time, the statement `<<` is used to move back to the start of the record.

```
   1.00 -111-222-111-222<·····················································
   2.00 -333-444-333-444<·····················································
   3.00 -555-666-555-666<·····················································
   4.00 ·····································································









·····························································0001.00:00001(00)
```

The result is that each line has been appended to itself.

10.19 R – Collect lines for multiple copying

The statement code R transfers the line numbers and work file numbers corresponding to the specified record to the EDT copy buffer so that the record can subsequently be copied multiple times using one of the statement codes A, B or O.

Statement code	Key
R	DUE or F2

As soon as one of the statement codes A, B or O is entered, the copy operation is executed, i.e. the records are inserted in the corresponding position. Unlike the statement codes C and M, R does not cause the content of the copy buffer to be deleted after the operation. It therefore remains available for further copy operations.

If, when R is specified, the copy buffer already contains entries that were generated using the statement codes C or M then the copy buffer is deleted before the specified line is entered and the message EDT0295 is output.

The copy buffer can also be filled with lines taken from different work files. The records determined by the copy buffer can then be copied into any work file.

The copy buffer contains the work file and line numbers of the records collected with R. The line numbers and content of the records that are to be copied should therefore not be modified between being collected with R and the performance of the copy operation with the statement codes A, B or O. However, if this does occur then the new contents are copied. If any records have been deleted in the interim then these are skipped without any warning during the copy operation.

Since the copy buffer only contains work file and line numbers, no conversion of the selected lines is performed during collection. Since the target character set is not determined until the target work file is specified, any conversion that is required is not performed until the statement codes A, B or O are evaluated (see section “Character sets” on page 47).

Note

If the screen is split then lines can be copied from the first to the second work window in a single dialog step. If items are copied from the second to the first work window then two dialog steps may be necessary depending on the processing sequence.

Example

```
   1.00 EDT is the BS2000  file<·····································
   2.00 editor, used for the user-<·····································
r  3.00 <·····································
   4.00 friendly creation and editing<·····································
   5.00 of BS2000 files in SAM and ISAM formats<·····································
   6.00 as well as text-like library<·····································
b  7.00 elements and POSIX files.<·····································
a  8.00 The repetitive operations which occur<·····································
   9.00 during editing, such as deleting<·····································
  10.00 ·····································
```

Line 3.00 is to be copied to a position before line 7.00 and after line 8.00. This is achieved by entering statement code R in line 3.00, statement code B in line 7.00 and statement code A in line 8.00.

```
   1.00 EDT is the BS2000 file<·····································
   2.00 editor, used for the user-<·····································
   3.00 <·····································
   4.00 friendly creation and editing·····································
   5.00 of BS2000 files in SAM and ISAM formats<·····································
   6.00 as well as text-like library<·····································
   6.10 <·····································
   7.00 elements and POSIX files.<·····································
   8.00 The repetitive operations which occur<·····································
   8.10 <·····································
   9.00 during editing, such as deleting<·····································
  10.00 ·····································


  21.00 ·····································
  22.00 ·····································
·····0001.00:00001(00)
```

Line 3.00 has been copied to positions before line 7.00 and after line 8.00.

10.20 S – Position the work window (horizontally and vertically)

S moves the work window to the required line and column position in two steps.

In the *first* step, i.e. after the statement code S has been transferred with DUE (F2 has the same effect as DUE in statement code S), the specified line is set to overwritable and is positioned in the second line in the work window. A column counter is displayed in the first line of the work window.

In the *second* step, the line below the column counter is used to move the position to the required column. This is achieved by entering blanks up to just before the required column position and then transferring the input. EDT then positions

– the line previously specified with S as the first line in the work window, and

– the work window to the first column which does not contain a blank in this line.

Statement code	Key
S	DUE or F2

The statement code S is ignored if it is entered in a line which is not (or is no longer) present. The statement code S is not permitted in EDIT-LONG mode.

If the line is not modified in the second step then no column positioning is performed. Any blanks or other characters that are entered do not modify the original line contents. If it is necessary to move the position to within a range of blanks then the user must enter a non-blank character at the corresponding location.

If the user overwrites the entire text of a line indicated with S with blanks, EDT positions the work window at the first column which is no longer visible on the screen.

In the second processing step (positioning at the required column), any other required statement codes can be entered (and combined). In this case, positioning is performed before the other statement codes are processed. This is of particular importance for the statement code O which is executed in accordance with the rules which apply to a column position which has been moved to the right.

Example

```
   1.00 SERNO.   ART.NO. ART.NAME       QUANTITY  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    CREAM          987       555<·························
   6.00 5        0091    SHAVING FOAM   350       30<··························
   7.00 6        0090    AFTERSHAVE     340       30<··························
   8.00 7        0092    RAZOR BLADES   200       30<··························
   9.00 ·····································································
```

The work window is to be positioned to line `4.00` and the column containing the product name. In the first step, `S` is therefore entered in line `4.00`.

```
       ----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
   4.00                PERFUME        400       60<··························
   5.00 4        0003    CREAM          987       555<·························
   6.00 5        0091    SHAVING FOAM   350       30<··························
   7.00 6        0090    AFTERSHAVE     340       30<··························
   8.00 7        0092    RAZOR BLADES   200       30<··························
   9.00 ·····································································
```

Line `4.00` has been set to overwritable. Blanks are entered in this line up to the required column position.

```
   4.00 PARFUME        400       60<·····································
   5.00 CREAM          987       555<····································
   6.00 SHAVING FOAM   350       30<····································
   7.00 AFTERSHAVE     340       30<····································
   8.00 RAZOR BLADES   200       30<····································
   9.00 ·····································································
```

After transfer, EDT has positioned the work window to line `4.00` and column 18.

Note

It is also possible to position at a line and column in a single dialog step using the @SETF statement. However, in this case, it is necessary to enter the column position as a number.

10.21 T – Syntax test by SDF

The statement code T passes the specified line – possibly together with continuation lines (as used by SDF) – to the SDF system component which verifies the command and statement syntax.

If the SDF option GUIDANCE=MIN|MED|MAX is set then the user is taken to a guided SDF correction dialog if incorrect SDF syntax is found.

If the user aborts the correction dialog or if no such dialog is possible then the line is displayed at the topmost window position (where it can be overwritten) and an error message is output as in the case of the @SDFTEST statement.

If the SDF syntax is correct or has been corrected then the text is entered in the work file. The format in which the statement is taken over depends on the SDF LOGGING option (see the description of the command /MODIFY-SDF-OPTIONS).

The current SDF settings apply. These can be modified with /MODIFY-SDF-OPTIONS.

Statement code	Key
T	DUE or F2

EDT distinguishes between 3 types of record content:

1. Records which start with (a single) '/' in column 1:

   These are checked for command syntax in accordance with the SDF syntax file hierarchy. Admissibility in terms of privileges or system environment is determined on the basis of the current user and the current environment.

2. Records which start with '//' :

   These are passed to SDF for statement verification. The external or internal program name must be preset using the statement @PAR SDF-PROGRAM as otherwise the statement code is rejected with the message EDT5320. The program name must also be known in a current SDF syntax file. Otherwise, the statement code is rejected with the message EDT5321.

3. Other data lines

   The specification of T is ignored for other data lines.

Lines that start with '/' or '//' and have the continuation character ('-') as their last character are chained with the continuation line if this also starts with '/' or '//' and the two lines are passed to SDF together when the statement code T is executed. It is not necessary to specify the statement code T in continuation lines. The specification in the first line is enough.

The verified command or statement overwrites the old command or statement in the work file together with all the continuation lines. If the command or statement is modified during verification by SDF (for example because LOGGING=INVARIANT was set in a preceding /MODIFY-SDF-OPTIONS command) then the affected lines are reformated and split into multiple continuation lines if necessary. The continuation character is set in the 72nd column.

The following lines are renumbered if necessary. Line numbers are assigned using the procedure “Insertion between two lines” (see section “Line number assignment” on page 36). If the lines generated by SDF cannot be inserted, the execution of the statement code is aborted with error message EDT5365.

If a verified command or statement contains errors and no successful correction dialog is conducted then EDT issues the message EDT4310 and aborts verification, i.e. any further selected commands and statements are not verified.

If the work file containing the lines that are to be checked has a character set other than EDF03IRV then it is necessary to take account of certain peculiarities when using SDF. In particular, characters which do not form part of the EBCDIC kernel are naturally only permitted in literals or comments. Furthermore, SDF always conducts the correction dialog in the character set defined for it with /MODIFY-TERMINAL-OPTIONS and also always interprets the byte sequences passed to it in this character set.

Consequently, if the currently defined GUIDANCE-MODE permits a correction dialog, EDT converts statements and commands into the character set defined in /MODIFY-TERMINAL-OPTIONS before passing them to SDF. If this fails, the statement code T is aborted with the message EDT5327.

If no correction dialog is possible, EDT performs conversion in accordance with other (less constraining) rules. If the work file possesses an EBCDIC character set then this is used without conversion. If the work file has an ISO character set then the corresponding EBCDIC reference character set is used. In all other cases, EDT uses the character set UTFE. If conversion is not possible then the statement code T is rejected with message EDT5453.

If SDF returns data then this is converted back into the work file's character set. If this is not possible then the statement code T is rejected with message EDT5453.

Note

If GUIDANCE=EXPERT then EDT simply displays errors reported by SDF in the form EDT4310. For a more precise error analysis, it is advisable to set GUIDANCE=MIN, MED or MAX.

Passwords and other operands that have been defined using OUTPUT=SECRET-PROMPT are replaced by P if the GUIDANCE setting is MIN, MED or MAX.

SDF does not detect invalid operands in ISP commands.

If /MODIFY-TERMINAL-OPTIONS has been used to define the character set UTFE then the screen layout in the SDF correction dialog is shifted if characters which are not present in EDF03IRV are used. This is due to the fact that SDF does not currently support Unicode. However, it does not generally lead to any functional limitations.

A command or statement must not exceed the maximum length of 16379 bytes either on input or output. Otherwise, the statement code T is aborted with the message EDT5325 or EDT5326.

Example 1

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

The EXPERT form of the non-guided dialog is set for SDF.

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

Lines 1.00 to 8.00 are to be checked by SDF for correct syntax.

```
   2.00 /MODIFY-FILE-ATTRIBUTES FILE-NAME=FILE2, -<·····························
   3.00 /                       NEW-NAME=FILE3, -<·····························
   4.00 /                       PROT=*PARAMETERS(UCCESS=*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 0002.0000
·····························································0002.00:00001(00)
```

The position is set to the first line of the invalid command and the command lines are output in overwritable form.

Example 2

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

The guided dialog with minimum help level is set for SDF.

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

Lines `1.00` to `8.00` are to be checked by SDF for correct syntax.

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


------------------------------------------------------------------------------
FILE-NAME             = FILE2
NEW-NAME              = FILE3
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 'UCCESS' COULD NOT BE IDENTIFIED
```

The user sees the guided SDF error dialog.

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


--------------------------------------------------------------------------------
FILE-NAME             = FILE2
NEW-NAME              = FILE3
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 'UCCESS' COULD NOT BE IDENTIFIED
```

The user corrects the error.

```
   1.00 /SET-JOB-STEP<·····················································
   2.00 /MODIFY-FILE-ATTRIBUTES FILE-NAME=FILE2,NEW-NAME=FILE3,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 2.00 to 4.00 are replaced by lines 2.00 to 3.00 and are reformated.

10.22 U – Convert lines into uppercase

The statement code U causes the specified record to be converted into uppercase. Conversion is performed in the same way as for @CONVERT TO=UPPER using the corresponding XHCS function.

Statement code	Key
U	DUE or F2

Example

```
   1.00 aaa bbb ccc ddd<·····················································
   2.00 eee fff ggg hhh<·····················································
   4.00 <····································································
u  5.00 iii jjj kkk lll<·····················································
```

Line 5.00 is to be converted into uppercase characters.

```
   1.00 aaa bbb ccc ddd<·····················································
   2.00 eee fff ggg hhh<·····················································
   4.00 <····································································
   5.00 III JJJ KKK LLL<·····················································
```

Line 5.00 has been converted into uppercase characters.

10.23 X – Modify lines

Statement code X sets lines to overwritable so that they can be modified.

Statement code	Key
X	DUE or F2

The changes take effect in the record section displayed in the data window. The ranges to the left and right of the displayed section remain unchanged. However, when inserting items with EFG it should be remembered that parts of records that are pushed out over the right-hand edge of the data window are lost. This can be avoided by using the statement code E for insertion.

The entire data window can be set to overwritable using F2 (see section "The F keys" on page 123). Specifying @PAR EDIT-FULL=ON causes EDT to set all the lines to permanently overwritable.

Note

The special function record marks 14 and 15 (see section "Record marks" on page 45) are deleted when a record is modified. A record with record mark 15 can only be modified (overwritten) if the write protection set with mark 15 is ignored by specifying @PAR PROTECTION=OFF. Conversion to uppercase or lowercase using the statement codes U or L or the joining of lines with statement code J are not considered to be modifications. In these cases, the record marks are retained.

Example

```
   1.00 BERGER   ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····················
x  2.00 DUCK     DONALD    WALTSTREET 8   DISNEYLAND<·························
   3.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   4.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································································
```

Line 2.00 is selected for modification.

```
   1.00 BERGER   ADALBERT  HOCHSTR.10     81234 MUENCHEN<·····················
   2.00 DUCK     DONALD    WALTSTREET 8   33333 DISNEYLAND<···················
   3.00 GROOT    GUNDULA   HAFERSTR.16    89123 AUGSBURG<·····················
   4.00 HOFER    LUDWIG    GANGGASSE 3A   80123 MUENCHEN<·····················
   5.00 STIWI    MANUELA   POSTWEG 3      80123 MUENCHEN<·····················
   6.00 ···············································································
```

The line that is to be modified is set to overwritable. The Zip code is inserted before DISNEYLAND.

10.24 1..9 – Insert lines

The statement codes 1..9 are used to provide new lines on the screen in front of the specified line. They can be filled with text and inserted in the work file after data transfer. For the distinction between new lines and empty lines (which correspond to records of length 0) see section “F mode” on page 101.

Statement code	Key
1..9	DUE or F2

Depending on the selected statement code (1..9), 1 to 9 new lines are provided before the specified line. The displayed area may be shifted in the data window in order to accommodate the new lines.
Line numbers are already formed for the new lines at this point using the procedure “Insertion between two lines” (see section “Line number assignment” on page 36). If insertion is not possible then the statement code is rejected with message EDT5365.
If no text is entered in one of the available new lines then no record is created in the work file.

Example

```
   1.00 1··9    INSERTING LINES<·····································
   2.00 <·····································································
   3.00 <·····································································
   4.00 1··9 can be used to insert lines in a work file.<·····················
   5.00 <·····································································
   6.00 111<···································································
3  7.00    222<·····································································
   8.00       333<·····································································
```

Three lines are to be inserted in front of line 7.00. This is achieved by entering the statement code 3 in line 7.00.

```
   1.00 1··9    INSERTING LINES<·····································
   2.00 <·····································································
   3.00 <·····································································
   4.00 1··9 can be used to insert lines in a work file.<·····················
   5.00 <·····································································
   6.00 111<···································································
   6.10 ·····································································
   6.20 ·····································································
   6.30 ·····································································
   7.00    222<·····································································
   8.00       333<·····································································
   9.00 ·····································································
```

The new lines 6.10 to 6.30 are provided.

10.25 1..9 – Set record mark

If one of the statement codes `1..9` is sent with the function key F3 then the corresponding record marks are set in the specified record (see section “Record marks” on page 45). These record marks can be used, for example, to position the data window or to copy the marked records (see @ON statement).

Statement code	Key
1..9	F3

The @ON statement (format 4) can also be used to set record marks.

11 Compatibility mode

This section describes compatibility mode and the way it interacts with Unicode mode.

The compatibility mode provides the full functionality of EDT V16.6B including the old L mode subroutine interface. The extended functions provided in Unicode mode are not available in compatibility mode.

In addition, in compatibility mode the functionality provided by EDT V16.6B has been extended in the following three modified or new statements.

11.1 @CODENAME – Define character set

In compatibility mode, the @CODENAME statement is used to define the character set for global use.

Operation	Operands	F mode, L mode
@CODENAME	[name] [,**F**ORCE={**Y**ES | **N**O}]	

name — Name of the character set that is to be defined. It must be known in XHCS.

If it is an 8-bit character set or the 7-bit character set `EDF03IRV` then this character set is used. In interactive mode, it is also necessary for the character set to be supported by the employed terminal. Otherwise, the statement is rejected with the message `EDT5259`.

If the character set is an ISO character set, a Unicode character set or a 7-bit character set other than `EDF03IRV` and if all the work files are empty and no file is open then processing switches automatically to Unicode mode in which the character set is defined for all the work files as in the case of the @MODE statement.

When operation switches to Unicode mode as a result of the @CODENAME statement, the mechanism for the automatic selection of the communications character set is also activated (see section "Communications character set" on page 53).

If the character set that is already defined is specified in the statement then the statement is ignored and no message is issued.

If `name` is not specified then – if permitted – EDT's default character set is used (see section "Character sets" on page 47).

FORCE=NO If `FORCE=NO` is specified or if the `FORCE` is not specified then it is only permissible to define the character set if all the work files are empty and no files are open for real processing.

FORCE=YES Specifying `FORCE=YES` causes the character set to be switched even if one or more files are not empty or files are open.

The data is not converted but is simply interpreted in the new character set. The display of the characters at the terminal changes accordingly. The new character set is also entered in the file's or library element's catalog entry if the work file is written back or an open file is closed.

Specifying `FORCE=YES` has no effect on ISO character sets, Unicode character sets or 7-bit character sets other than `EDF03IRV`.

11.2 @IF (format 5) – Query EDT parameter settings

This format of the @IF statement can be used in EDT procedures or in L mode to query the operating mode that is currently set (see section "Introduction to the EDT operating modes" on page 21). Depending on the result, a specified string either is or is not processed as input.

Operation	Operands	L mode
@IF	**OP**ERATING-MODE = { **U**NICODE / **C**OMPATIBL }	:[text]

OPERATING-MODE=

The EDT operating mode is checked.

UNICODE The condition is fulfilled if EDT is in Unicode mode.

COMPATIBLE

The condition is fulfilled if EDT is in compatibility mode.

text EDT statement or data line. If the condition is fulfilled, the string is treated as if it had been entered at the prompt in L mode. In particular, the decision to interpret the text as data input or as a statement is made in accordance with the same rules (for more information, see section "L mode" on page 126).

The `text` operand starts immediately after the character `':'`, i.e. any specified blanks form part of the operand and are taken over into the line in the case of data input.

If `text` is not specified, the statement is ignored.

11.3 @MODE – Change operating mode

The @MODE statement is used to switch between the operating modes (compatibility mode and Unicode mode).

Operation	Operands	F mode, L mode
@MODE	**OP**ERATING-MODE = { **U**NICODE / **C**OMPATIBL }	

OPERATING-MODE=
The EDT operating mode is changed.

UNICODE EDT changes from compatibility mode to Unicode mode. If EDT is already running in Unicode mode, the statement is ignored.

COMPATIBLE
EDT changes from Unicode mode to compatibility mode. If EDT is already running in compatibility mode, the statement is ignored.

It is only possible to change operating mode if all the EDT work files are empty and no files are open. Otherwise, the statement is rejected with the message EDT4983.

Changing the operating mode amounts to terminating EDT in one mode and then restarting it in another mode. When this is done, all the settings are lost and all the variables are reinitialized. For further details, see section “Activating compatibility and Unicode mode” on page 615.

Alongside an explicit change of operating mode using the @MODE statement, an implicit change from compatibility mode to Unicode is also possible if all the work files are empty, no files are open and a @CODENAME statement specifying an ISO character set, a Unicode character set or 7-bit character set other than EDF03IRV is issued. However, an implicit change from Unicode mode to compatibility mode never occurs.

11.4 Activating compatibility and Unicode mode

If EDT is started in interactive or batch mode with /START-EDT then compatibility mode is initially activated. It is possible to start EDT directly in Unicode mode by issuing the command /START-EDTU (see also section "Starting EDT" on page 87). If EDT is started via one of the subroutine interfaces then the rules summarized in the table below apply.

An automatic change from compatibility to Unicode mode occurs during the EDT initialization phase if

– EDT was called in interactive mode and a Unicode character set has been defined for the terminal with/MODIFY-TERMINAL-OPTIONS,

– EDT was called in a procedure or a batch operation and the character set of the file assigned to SYSDTA (or of the library element) is a Unicode or ISO character set,

– EDT finds an EDT start procedure which is coded in a Unicode or ISO character set.

A subsequent change from compatibility to Unicode mode is possible if

– the @MODE or @CODENAME statement with the corresponding operands is applied to empty work files,

A change from Unicode mode to compatibility mode occurs if

– a @MODE statement with the corresponding operands is applied to empty work files;

For further details, and in particular for the description of the operands and their default values, see the description of the @MODE statement.

Since it is only possible to change between Unicode and compatibility mode if all the work files are empty no data loss or corruption is possible in files when the mode is changed. The change is therefore global in nature and no confirmation query is issued.

In F mode, the currently set operating mode is clear from the layout of the status display in the statement line (see section "F mode" on page 101). In addition, both in Unicode mode and compatibility mode, the current operating mode is output when @STATUS=ALL is issued (this information is also available in L mode).
Finally, the @IF statement can be used in procedures to determine the current operating mode.

Because a change of operating mode is only possible when the work files are empty, it is not possible to exchange data between the operating modes in this way. In addition, all the other global data such as the contents of variables or parameter settings specified with @PAR are not taken over in the other mode and are therefore lost.
Instead, EDT acts as if the switch were made between different instances of the program which do not know anything about one another.

Consequently, the changeover is equivalent to exiting EDT in one mode and then restarting it in the other mode. The following steps are performed:

– In the old mode, the string variables #S00 to #S20 are exported to the corresponding S variables SYSEDT-S00 to SYSEDT-S20 and are then reinitialized in the new mode provided that the S variables exist and are of type STRING.

– The EDT start procedure is executed in the new mode (provided that the prerequisites for this are fulfilled in accordance with section "EDT start procedure" on page 72).

– In addition, all the EDT parameter settings are reset to their initial values, i.e. even if the user changes several times between compatibility and Unicode mode, EDT does not remember any previous settings.

11.5 Subroutine interfaces and operating modes

On the one hand, it is possible to load and initialize EDT via the subroutine interfaces and, on the other, the subroutine interfaces can be called in an already initialized EDT environment (for example, after F mode has been switched to with @DIALOG and the user has issued @HALT to return to the calling program).
If, in the latter case, the work files are not empty, then EDT is either in compatibility or Unicode mode and no changeover is possible.
It is permissible to change operating mode directly after initialization or when all the work files are empty.
It is necessary to distinguish between these cases when issuing calls at the subroutine interface.
The following table provides an overview of which subroutine interfaces are permitted in which environment and how the calls are handled.
It also considers the case in which only EDT V16 is available in the current system environment.

	EDT V17 not present	**EDT V17 present, switchover permitted and call via IEDTCMD**	**EDT V17 present, compatibility mode active, switchover prohibited**	**EDT V17 present, Unicode mode active, switchover prohibited**
Call via IEDTGLE V16 interface	As previously	Set compatibility mode	No special action necessary	Convert to V17 interface – if not possible: error
Call via IEDTGLE V17 interface (compatible format)	Convert to V16 interface	Set Unicode mode	Convert to V16 interface	No special action necessary
Call via IEDTGLE V17 interface (extended format)	Error	Set Unicode mode	Error	No special action necessary
Call via L mode interface	As previously	Set compatibility mode	No special action necessary	Error

The following functions are available depending on the set EDT V17.0A mode and the employed subroutine interface:

L mode interface:

All the functions present in compatibility mode are available. If Unicode mode has been activated, all calls via the L mode interface are rejected with RC `X'0C'`. Since it is usually only permitted to switch to Unicode mode and back again if the work files are empty, it is not possible to access the new Unicode mode record structure via the L mode interface.

IEDTGLE interface (V16 format):

All the functions present in compatibility mode are available. If Unicode mode is active then `IEDTCMD`, `IEDTEXE`, `IEDTDEL` and `IEDTREN` can be used without restrictions. `IEDTPUT`, `IEDTGET` are permitted in Move mode. However, conflicts may occur if the buffer size is inadequate. In the past, the provided buffer was sometimes too small to accept the record. In this case, a corresponding return code was supplied and this still occurs.

IEDTGLE interface (compatible V17 format):

Only those Unicode mode functions are provided that can be converted to equivalent V16 format functions. This format is intended for users who want to use the new interface but who need to make sure that their programs will still run with EDT V16.6B. Locate mode is no longer provided.

IEDTGLE interface (extended V17 format):

Only the Unicode mode functions are available. Locate mode is no longer provided.

Detailed information about the subroutine interfaces is provided in the manual “EDT Subroutine Interfaces” [1].

11.6 Character sets

EDT makes it possible to process texts present in different character sets. The method of operation in compatibility mode is very different from in Unicode mode. Since EDT never converts data from one character set to another in compatibility mode only limited support is available in this mode.

In compatibility mode, EDT always has precisely one defined character set.
Texts can therefore only be processed simultaneously if they are present in the same character set. This character set can only be modified if no data is present in the work files. It is therefore defined when data enters a work file. EDT also uses this character set to communicate with the terminal or to read and write its input/output when communicating with other sources. In particular, EDT can only process files if it is guaranteed that these can be fully displayed on screen. Certain character sets are therefore excluded, for example `EDF03DRV`, `EDF046` and ISO character sets.

11.6.1 Supported character sets

EDT only ever permits character sets which are supported by the current XHCS installation. The necessary conversions are handled via XHCS and the properties of the characters (uppercase/lowercase, special characters) are provided by XHCS.

In compatibility mode, there are further restrictions due to VTSU. The only 7-bit character set permitted by EDT is `EDF03IRV` even if others, for example e.g. `EDF03DRV`, are defined in XHCS since `EDF03IRV` is the only 7-bit character set that can be used to communicate with terminals.
This restriction also applies in batch mode. Furthermore, only EBCDIC character sets are permitted, not ISO character sets. If EDT communicates with a terminal then the character set must be compatible with it. In contrast, all the 8-bit EBCDIC character sets are permitted in batch mode.

EDT accepts all input even if the characters/bytes do not form part of the recognized code. All bytes with codes which are not present in the current character set are displayed as smudge characters at the terminal.
They are ignored during lowercase/uppercase conversion and should not be used as special characters such as tab characters (however, there is no mechanism to prevent this). However, they are not (usually) modified by EDT.

Unicode character sets and ISO character sets are not permitted in compatibility mode.

11.6.2 Strings

All strings are always interpreted and processed in a character set.

In compatibility mode, there is precisely one character set in which data is read, stored, displayed, processed and written. This is always the **current** character set.

11.6.3 Communications character set

The communications character set is the character set used by EDT to exchange data with a terminal.

In compatibility mode, this is the character set currently defined in EDT. There is therefore no independent communications character set.

11.6.4 Character sets in work files

The handling of character sets in compatibility mode is very different from in Unicode mode.

In compatibility mode, EDT always has precisely one defined character set. On start up, it determines a default value for the character set.
If EDT reads its input from a terminal, it takes the character set from the terminal option `CODED-CHARACTER-SET`. If `7-BIT` is specified here, it uses `EDF03IRV`, otherwise the character set specified here.
If EDT reads its input from `SYSDTA`, it uses the character set which is assigned to `SYSDTA`. If no character set is specified here `(*NONE)`, it uses `EDF03IRV`.

On start-up, the current character set is initially the default character set. It can be modified as long as there is no data in the work files. As soon as the first record is written to a work file, the character set is fixed and cannot be modified until all the work files are empty again. The change can be performed implicitly by reading a file or explicitly with the @CODENAME statement.

The @CODENAME statement is only executed if the character set is permitted and if no data is present in the work files. Otherwise the @CODENAME statement is either rejected or has no effect.

The @MODE statement can be used to switch to Unicode mode (see @MODE statement).

Note

The @CODENAME character set can be used to modify the current character set but not to define a character set. The character set may therefore be changed when a file is read in.

11.6.5 Reading in files

When reading in a file, EDT evaluates the character set of the catalog entry.

In compatibility mode, the file is only read if this character set is identical with EDT's current character set or if no data is present in the work files. The corresponding character set is then defined.
If data with a different character set is present in any of the other character sets then the file is not read in. The file is also not read if the character set is not supported, i.e. if in interactive mode, for example, it cannot be displayed at the terminal.

11.6.6 Writing files

When new files are written, the character set of the work file is written to the catalog.

In compatibility mode, when a file is closed after being written, the work file's character set, i.e. the current EDT character set, is entered in the catalog.
The exception to this rule is that EDT does not modify the catalog entry if the file has the character set `*NONE` and the current character set is `EDF03IRV`. In this case, the value `*NONE` is retained.

11.6.7 Copying between work files

The @COPY statement, @MOVE statement, @ON statement or statement codes can be used to copy data from one work file to another.

In compatibility mode, only one character set is involved in this operation with the result that data is not converted but is transferred unchanged.

11.6.8 Character set in statements

In compatibility mode, although statements are read using the current character set, the statements are analyzed in EDF041. Consequently, a byte with X'7C' is always expected as the EDT statement symbol (unless it has been redefined).
If the statement is read in EDF04DRV then the character '§' must be used as the EDT statement symbol instead of '@' since this character is coded X'7C'.
The same applies if symbols are redefined. For example, if the currency symbol '¤' has been defined as the EDT statement symbol in EDF041 then the Euro character '€' must be used in EDF04F.
Finally, it is also possible to define a character as the statement symbol which is not even a special character in another character set. For example, after @CODENAME EDF04DRV; §:|; @CODENAME EDF041, the statement symbol is the character 'ä'. Since this is not a special symbol, no statements at all are recognized in L mode.

Note

This situation is remedied in Unicode mode in which statements are always analyzed in UTFE.

11.6.9 String variables

String variables may be assigned any text such as a line in a work file. They can be accessed globally across work files. Every string variable has a content at all times since it is assigned the character '␣' (X'40') on initialization.

In compatibility mode, string variables do not have a character set. Their contents are always interpreted in the current character set.
Even after its content has been deleted or it has been assigned an empty string, a string variable is always initialized with the character '␣' (X'40').

Note

Workarounds (save data in string variables, delete all work files, insert from the string variables) can be used in compatibility mode to change the character set between saving and inserting a string. In this case, the data to be copied is of course transferred unchanged. The result is not usually meaningful and will certainly be different from in Unicode mode. For example, after the sequence

```
@CODENAME EDF047
@CREATE 1 ' πλάτων '
@CREATE #s0:1
@DELETE
@CODENAME EDF041
@CREATE 1 #S0
```

line 1 of a work file in compatibility mode has the content **Ðëáôùí**.

In Unicode mode, EDT attempts to convert the string ' **πλάτων** ' into EDF041. This is either rejected or the substitute character is used.

11.6.10 S variables and job variables

The @GETVAR, @GETLIST and @GETJV statements can be used to transfer the contents of S variables or job variables to string variables or work files.
In compatibility mode, the content is taken over unchanged, i.e. the character set is ignored.
The @SETVAR, @SETLIST and @SETJV statements can be used to create S variables or job variables and assign them a value.
In compatibility mode, the content is assigned unchanged, i.e. the character set is ignored.

11.6.11 POSIX files

POSIX files also record no information about the associated character set.

In compatibility mode, it is possible in the @XOPEN, @XCOPY and @XWRITE statements, to use the CODE operand to specify whether the file is an EBCDIC (CODE=EBCDIC) or ASCII (CODE=ISO) file.

In the latter case, when it is read, the text is converted using a fixed table ISO88591 -> EDF041, and when written it is converted using a fixed table EDF041-> ISO88591. The @PAR CODE=EBCDIC/ISO statement can be used to define a default setting.

11.7 Starting EDT

In compatibility mode, EDT is loaded and started with the command /START−EDT. The /START−EDT command corresponds to the command in EDT V16.6B. For a description, see the manual “EDT V16.6B Statements” [2].

For reasons of compatibility, it is still possible to call EDT using /START−PROGRAM. EDT is then loaded as a main program with one of the following BS2000 commands and is started in compatibility mode:

Command	AMODE
START-PROGRAM $.EDT	AMODE 31
START-PROGRAM *MODULE ($.SYSLNK.EDT.170, EDTC, RUN-MODE=*ADVANCED)	AMODE 24

12 Migration aids

To assist you in migrating between versions, this chapter presents the differences between compatibility mode and Unicode mode

12.1 Compatibility mode

There are two new statements in compatibility mode.

On the one hand, the @MODE statement has been introduced to make it possible to switch to Unicode mode.

On the other, format 5 of the @IF statement has been introduced to enable users to query the current operating mode and react appropriately.

In addition, a number of errors have been corrected in compatibility mode. These corrections may result in changes in system behavior.

12.2 Unicode mode

Unicode mode, which solely incorporates the most important extensions to EDT V17.0A such as support for Unicode character sets and long records, operates differently from compatibility mode when performing a wide variety of functions. The general principle is that the constraints applying to modifications in L mode are significantly greater than in F mode since the potential consequences in interactive mode are considerably less serious.

The present chapter provides a brief overview of the differences between Unicode mode and compatibility mode. It is particularly important to take account of these notes when procedures have to be migrated to Unicode mode. This will not normally be possible without modifications which may sometimes be time-consuming.

The presentation is subdivided into the following subsections:

- EDT V16.6B functions that are no longer supported in EDT V17.0 Unicode mode.
- Statements which act differently in the EDT V17.0A Unicode mode than in EDT V16.6B.
- Changes in the screen display and input/output in EDT V17.0A compared to EDT V16.6B.
- Changes in the general or work file-specific settings in the EDT V17.0A Unicode mode compared to EDT V16.6B.
- Changes to the subroutine interface if the new V17 format of the interface is to be used or if EDT V17.0A's old V16 format is to be used in Unicode mode.

During implementation, a large number of errors were identified in EDT V16.6B. Many of these have only been corrected in Unicode mode. Not all of these corrections are listed here.

12.2.1 Functions that are no longer supported

Compatible L mode syntax checking for EDT V15 statements is no longer supported. There is no longer a `SECURITY` operand in the @SYNTAX statement. This syntax analysis which was set by default when procedures were used was unreliable and accepted a large number of syntactically incorrect entries. As a result, however, it is necessary to correct these specifications when switching over to Unicode mode.

The old L mode subroutine interface is no longer supported since the EDT internal record format was revealed at this interface. Only the more recent `IEDTGLE` interface is now available as a subroutine interface.

The old @RUN statement for calling a user program, in the form in which it was defined in EDT V16.6B, no longer exists in Unicode mode since the EDT internal record format was also revealed at this interface. It has been replaced by a new @RUN statement. Alternatively, the @USE statement is also available.

Locate mode during data transfer via the `IEDTGLE` interface is no longer supported. The internal EDT record format was also revealed at this interface. It is now only possible to use MOVE mode.

Terminals which use Arabic or Farsi character sets are no longer supported. These special terminals are not suitable for the use of Unicode character sets. If necessary, they can be used in compatibility mode.

The 3270 terminal (IBM) is no longer supported. This special terminal is not suitable for the use of Unicode character sets. If necessary, it can be used in compatibility mode.

The @CODE statement is no longer supported. It is no longer required if character sets for files are used correctly.

The @UPDATE statement is no longer supported since the introduction of the F mode means that it is no longer needed.

The @ZERO-RECORDS statement is no longer supported. It is no longer required due to the new procedure for handling empty records.

The direct specification of a list variable as an output medium is no longer possible in the @LOG statement. It is, however, possible to use the `/ASSIGN-SYSLST` command to assign a list variable to the system file `SYSLST`.

12.2.2 Modified statement actions

The action of a number of statements has been modified in special cases. In most cases, the reason is that the old mode of action was inconsistent and led to unexpected results. In many cases, the change corresponds more accurately to a patch.

12.2.2.1 I/O statements

When a file is read (@OPEN, @COPY, @XOPEN, @XCOPY, @READ, @GET statements), empty lines (lines of length 0) are treated as normal lines. In the past, empty lines were eliminated on reading and were therefore lost when files were written back.

In the @OPEN statement (format 1), `TYPE=CATALOG` is the default setting for existing DMS files. In the past, this setting was `TYPE=SAM`.

When an empty work file is written or written back, an empty file is created.

When writing or writing back to a file which has the file attribute `SECONDARY-ALLOCATION=0`, the attribute is evaluated, i.e. if necessary, the file is not extended and an error message is output.

If a file is open in a work file then the @WRITE statement without any operands is no longer rejected but is interpreted as @WRITE (format 1) and the file is written back.

The attempt to close a file (@CLOSE statement) or write back an existing file (@WRITE statement) may be rejected if the file has a character set other than that defined for the work file and the `CODE` has not been specified. In EDT V16.6B, this situation could only occur if the work file had been deleted and then reconstructed in another character set. In such cases, the file's character set was modified without any query being issued.

In the @XOPEN and @XWRITE statements, `MODE=REPLACE` now has the same effect as `MODE=NEW` if the file does not yet exist. In the past, the statements were rejected. The response is now the same as for the @OPEN and @WRITE statements.

The @READ and @GET statements can now be used to read files with nonstandard attributes.

12.2.2.2 Work file statements

When a work file is completely deleted with the @DELETE statement issued without parameters, any file open in the work file or any opened library element is implicitly closed. The work file is then no longer in use (as with @DROP). In the past, the file remained open and was empty following a subsequent @CLOSE statement. If this action is genuinely required, it is necessary to delete all the records by specifying a range.

The @DROP statement executes an implicit delete @DELETE, i.e. if a file is open then it is also closed. In the past, the file remained open but could no longer be accessed.

The syntax of the @COPY (format 2) and @MOVE statements has been changed with the result that the specification of a source work file is no longer accepted when string variables are copied or moved. Since string variables do not belong to any particular work file, this specification was both meaningless and confusing.

In the @COPY (format 2) and @MOVE statements, it is possible to specify multiple comma-separated send ranges each of which are associated with multiple receive ranges.

12.2.2.3 ON statements

Case sensitivity in searches using @ON (which can be defined using the @SEARCH-OPTION statement) now applies independently of the @PAR LOWER setting. It also applies to the @ON formats 8 to 10. The behavior in the past had no justification and was confusing since @PAR LOWER simply determines the conversion of characters on input from the terminal and the display of lowercase characters in the work window.

In the past, if the `MARK` operand was not specified in the @ON statement (format 3) then hit lines were not marked in L mode. Now they are marked with `1` as in F mode. The previous behavior was inconsistent: there should be no subtle differences between L mode and F mode.

If a backward search is performed with @ON (`R` operand), then if a hit is found the position is now moved far enough to the left to prevent any further hit from extending into the preceding one. The previous procedure could sometimes lead to backward searches indicating different hits from forward searches.

No column range is now specified in the @RANGE statement. Since this column range had nothing to do with the range symbol that was defined with @RANGE and was only taken into account in the @ON statement, it has now been transferred to the @SEARCH-OPTION statement.

12.2.2.4 Tabulators

When defining software tabulators using the @TABS statement (format 2), it is now necessary to specify at least one tab position in addition to the tab character. The tab positions must be sorted in ascending order. In the past, it was possible to issue a @TABS statement without a tab position. However, this had no effect and not even the tab character was modified.

Hardware and software tabulators can be defined independently of one another. It is possible to activate either of them.

In the display function provided by the @TABS statement, it is now also possible to differentiate between hardware and software tabulators. The @TABS VALUES statement outputs the positions of the hardware tabulators, while the @TABS ::VALUES statement outputs the positions of the software tabulators together with the defined tab character. In the past, the positions of the hardware tabulators (if defined) were prioritized on output. It was then not possible to display the positions of the software tabulator.

12.2.2.5 Miscellaneous

The `ISO` operand in the @INPUT statement (format 3) is ignored. Hexadecimal input is based on the coding of the current work file's character set.

The output format of the @STATUS statement has been modified since some of the items in the output are no longer relevant, some global properties are now work file-specific and there are other properties that need to be output.

In the @VTCSET=OFF statement, only those line mode control characters that do not belong to the `EBCDIC-DF03` kernel are converted into smudge characters. This contributes to the purpose of the statement which is to prevent the fragmentation of the screen output due to control characters.

No line count is now specified for screen output in the @VDT statement. Screen output with the @PRINT V or @PRINT E statement now always uses the number of lines that can be displayed on the screen (in L mode this is always 23 lines).

When the @VDT F2 statement is issued, the position is no longer moved back to column 1. Instead, the screen remains at the specified start column.

The @VDT statement now also supports the additional formats F3 and F4.

The statements @COMPARE, @FSTAT, @SHIH, @SHOW, @STAJV and @STATUS output information to work file 9. If a file is already open there then the output is rejected. In the past, this was only the case for the @SHIH and @SHOW LIBRARY statements.

The characters defined as delimiter characters using the @QUOTE statement must not be the same as the wildcard characters (see the @SYMBOLS statement). Although, in the past, the @SYMBOLS statement was rejected if a delimiter character was to be defined as

a wildcard character, the @QUOTE statement was nevertheless accepted if a wildcard was to be defined as a delimiter character. In some cases, this could lead to problems during the analysis of statement syntax.

The special marks (marks 13, 14 and 15) are no longer deleted by the @DELETE MARK statement. Even though, in the past, the special marks could only be set via the subroutine interface they were not protected against deletion by ordinary EDT users. This is the behavior described in the V16.6B manual.

Column number specifications in the @SEPARATE statement now operate recursively, i.e. if separating a record creates a continuation record which exceeds the specified column number, then the continuation record itself is split. In the past, the continuation record was not separated, thus obviating the point of specifying a column number as a separation point.

Specifying a separator in @SEPARATE may result in the creation of records of length 0 if multiple separators occur in immediate succession in the record. This change is the result of the desire to make the handling of empty lines consistent.

It is now also possible in F mode to save the last defined current line number and current increment in the @SET statement (format 6) and to restore the saved settings. In the past, this function was only effective in L mode.

A new @RUN statement with modified syntax and a modified interface has been introduced.

The @SHOW statement (format 1) no longer uses the `ISO4` operand. The layout of the information line has been adapted.

In batch mode, the @DIALOG statement is rejected with the message EDT5400. In compatibility mode and in EDT V16.6B it is ignored without any message if switch 5 is set. If switch 5 is not set, it is rejected with message EDT5400 in compatibility mode and with the (inappropriate) message EDT5409 in EDT V16.6B.

In the @SDFTEST statement, the explicit specification of either the name of the program whose statements are to be checked or of the name type applies only to the current call and does not implicitly modify the settings made for these values using @PAR SDF-PROGRAM or @PAR SDF-NAME-TYPE.

In the @SDFTEST statement, it is now possible to check commands and statements with more than 255 continuation lines (up to the maximum buffer length of 16379 bytes).

A record marked by means of the statement code `S` is always positioned in the second screen line. If necessary, the information line and tab line are temporarily hidden.

If a statement which is retrieved into the statement line using # contains characters which cannot be displayed in the current communications character set then `?`s are output instead of these characters.

12.2.3 Changes in the screen display and on input/output

In the status display in the EDT work window (right-hand part of the statement line), the column number display now has five digits and the work file number display two digits. This means that the maximum possible length of the input in the statement line is reduced by three characters. This change is necessary due to the support now provided for long records and for all 23 work files in F mode.

The end of a record in the data window is now indicated by the terminal-specific character [LZE] in the data window. The terminal fills the remainder of the screen to the right of [LZE] with protected NULL characters (X'00'). No further input is possible in this area. Either the input must be made in front of the [LZE] or the [LZE] must be overwritten. This change is the result of the desire to make the handling of empty lines (records of length 0) consistent.

The characters in the remainder of the screen line after [LZE] can no longer be displayed using characters other than those specified for the terminal at the hardware level (normally NULL). The EDT statement @SYMBOLS FILLER='char' can therefore no longer be used in Unicode mode to define the filler character displayed between the end of the record and the end of the screen line in F mode. This change is the result of the desire to make the handling of empty lines (records of length 0) consistent.

The function keys [K4] to [K15] are now handled in the same way as [K3] for the purposes of data transfer in F mode. In the past, EDT's behavior when these keys were entered was undefined.

In L mode, an empty input with the [F1] key creates an empty line (record of length 0). In the past, [F1] was treated in the same way as [DUE]. This change is the result of the desire to make the handling of empty lines (records of length 0) consistent.

The column counter, tabulator display or information line are only displayed in a work window if the work window is large enough to display at least one line of data. If it is not then all or some of the additional displays are hidden. In the past, it was possible to reduce the window to such an extent that the additional information was visible even though no data lines could be seen.

The information line (@PAR INFO=ON) has a new layout.

If a work window has so few lines that it is not possible to display a complete data line in hexadecimal mode then EDT does not switch to hexadecimal mode or, if hexadecimal mode was previously activated, then it is deactivated. Since the display of a data line coded in Unicode may require up to 6 hex lines, it is only possible to work sensibly with hex mode if a sufficiently large work window is available.

In hexadecimal mode, only hexadecimal digits (0...9), A...F) and NULL characters are displayed in the hex lines and these, together with blanks, are also the only values permitted for input. Null bytes at the end of a record are not removed and records which are

still empty after the end of the file and which are still displayed in the data window are also displayed by means of NULL characters in the hex lines. The previous procedure was inconsistent and was also not documented.

The error dialog in hexadecimal mode has been modified. If the error is not corrected, the dialog is aborted and the incorrect specifications are discarded.

In test mode, all output is now uniformly sent to SYSLST. In the past, although normal logging entries were written to SYSLST, any output marked as incorrect was sent to SYSOUT.

When output is written to SYSLST, the feed control character X'41' is replaced by X'40' and an additional empty line is written in front of the line in question if SYSLST has the character set UTFE. This change was necessary to permit the generation of interpretable 1-byte feed control characters in SYSLST files encoded in UTFE.

If line feeds extend over page boundaries then no additional empty lines are generated on the new page. The other line feeds are rejected.

If a library directory is output using the @SHOW statement (format 1) then the header line is no longer output in the first line of the work file.

12.2.4 Changes in the general or work file-specific parameter settings

12.2.4.1 Character sets

Each work file may have its own character set. This one of the most important new features of the EDT V17.0A Unicode mode.

When EDT starts and a work file has been completely deleted, no character set is any longer defined for the work file (*NONE). This change is due to the fact that a separate character set is now possible for each work file.

When data is read into a work file, it is converted into the work file's character set if this is not *NONE. An implicit change of character set now only occurs if the work file's character set is *NONE. This change is due to the fact that a separate character set is now possible for each work file.

When EDT starts, @PAR LOWER=ON is set for all the work files. It is no longer possible to instruct editors which support Unicode character sets to start in uppercase mode.

12.2.4.2 Line numbers

There is now only one current increment. This can be modified both with @SET (format 6) and with @PAR INCREMENT. In the past, there were two different increments which were used with different statements even though this was not clearly documented.

A number of statements cause lines to be inserted. If it is not possible to insert a set of lines at an increment of `0.01` then the insert operation is no longer aborted. Instead, EDT attempts the insertion process again down to the smallest possible increment of `0.0001`. There was no justification for the abort of the operation in the past.

In the case of statements which write their output to empty (or implicitly deleted) work files, e.g. @COMPARE or @SHOW, line numbers are assigned using the same procedure as for the @COPY statement, i.e., if necessary the increment is reduced and the lines are renumbered in order to permit the output of all the lines. The previous procedure was inconsistent and was therefore confusing.

12.2.4.3 Work file-specific

There is now a work file-specific symbolic line number `?`.

The condition, queried with @IF (format 3), of whether a hit was identified the last time the @ON statement was run is now also recorded on a work file-specific basis as is the corresponding column number.

12.2.4.4 Miscellaneous

It is no longer permissible to use any special characters for the special character operand type (`spec`). Instead, only the special characters explicitly listed in the operand description may be used. This change is necessary because the special characters have a special significance during syntax analysis and exotic multibyte special characters from Unicode character sets can cause problems during this operation.

The program name for the SDF syntax check, the type of program name for the SDF syntax check and the characters for separating records are now defined for each work file separately and not globally. This change helps simplify the behavior of the @PAR statement. There are now only two exceptions to the rule that @PAR can be used to make settings for each work file separately.

The initial value of the symbolic line number for hit lines `?` is now `0.0000` instead of `0.0001`. By predefining an invalid value, it is possible to determine whether any search statement has been issued.

The special mark 13 (record should be ignored when the file is written) is now also taken into account for POSIX files. The previous exceptional treatment of POSIX files was unjustified.

Specifications of file names, library elements and job variable names in statements are subjected to a syntax check and may be rejected with a syntax error message if appropriate.

All interruptible statements can now be aborted with K2 and /INFORM-PROGRAM. In the past, behavior was inconsistent.

The previous handling of European 7-bit terminals is no longer supported since it does not harmonize with the general support for 7-bit character sets. Special handling has been implemented to support the character set EDF03DRV, the only national 7-bit code recognized in XHCS.

12.2.5 Changes to the subroutine instance

This section describes the changes that users must note – in addition to the elimination of the L mode interface and the replacement of the @RUN interface (see section "Functions that are no longer supported" on page 624) – if they want to use the IEDTGLE interface in their programs.

Here, it is necessary to distinguish between use of the V16 format of the IEDTGLE interface (old macros or new macros with a corresponding VERSION parameter), the V17 format of the IEDTGLE interface (new macros with corresponding VERSION parameter) or the compatible V17 format (can be controlled using the flag EGLCOMP). The compatible V17 format should be used if the application needs to run in environments in which only an EDT version lower than V17.0A is installed.

If read or write operations are called in Locate mode in V16 format while EDT is in Unicode mode and it is not possible to switch to compatibility mode (e.g. because not all the work files are empty) then the call is rejected with a return code.

When the global status is read (IEDTGET function with pseudo work file 'G') in V16 format then the field for the globally specified character set which is omitted in V17 format is only set to a value other than 'blank' if the same character set is specified in all the non-empty work files.

In V16 format, strings that are transferred in the buffer EDTREC are interpreted in the character set defined for the relevant work file. V16 format functions that use the COMMAND, MESSAGE1 and MESSAGE2 buffers can only be processed correctly in Unicode mode by switching to compatibility mode or if the same character set is defined for all the work files (see the @CODENAME GLOBAL,... statement). Otherwise, the relevant function is rejected with a return code.

In both formats, the calling program must take account of the fact that in Unicode mode records longer than 256 bytes are supplied and that long records are truncated. If insufficient buffer space has been made available, then it is necessary, at the very least, to evaluate the return code (EAMAC04) and react accordingly.

In both formats, the calling program must take account of the fact that in Unicode mode records of length 0 are supplied.

In both formats, the IEDTINF function always returns the value 0 in the EGLINFM field (number of memory pages required for the static data area) of the EDTGLCB control block.

The flag EGLREOR (suppress memory reorganization) is no longer present in the V17 format of the EDTGLCB control block. New additions to this control block are the fields EGLCCSN (name of the character set in which the buffers COMMAND, MESSAGE1, MESSAGE2 and EDTREC are coded) and EGLCOMP (flag for compatible V17 format), the field EGLIND2 (display that EDT is running in compatibility mode) and the return code EUPCMPER (use of incompatible functions in the compatible format). In the compatible V17 format, the field EGLCCSN may only contain blanks.

In the V17 format of the control block EDTUPCB the flag EUPNUNI (block switch from Unicode to compatibility mode) is new. In compatible V17 format, EUPNUNI must not be set.

The V17 format of the EDTAMCB control block no longer contains the field EAMMMODB and the two flags EAMMOVM and EAMLOCM (all used in connection with MOVE mode). In addition, the equates for the unused marks (EAMMK10, EAMMK11, EAMMK12 and EAMMK0) are no longer present.

The field EPGCCSN (name of the globally defined character set in EDT) is no longer present in the V17 format of the EDTPARG control block. This is now present locally for each work file as the name EPLCCSN in the EDTPARL control block.

The field EPLCCSN is new in the V17 format of the EDTPARL control block (this field was previously present with the name EPGCCSN in the EDTPARG control block). In addition, the field EPLCCSNG (character set applies locally for all work files) is new while the fields EPLSTCOD (code default: EBCDIC/ISO) and EPLOPNXC (code of POSIX file: EBCDIC/ISO) are no longer present.

If it is necessary to use external statement routines (user statements defined with the @USE statement) which are implemented to use the V17 format of the subroutine interface then the developer of the statement routines must also provide an initialization routine. The presence of the initialization routine indicates to EDT that the V17 format is understood.

13 Messages

The EDT messages are provided by the BS2000 component MIP (*Message Improvement Processing*). When a message is output it may contain more recent text passages (*so-called inserts*).
The user can define the scope (/MODIFY-JOB-OPTIONS INFORMATION-LEVEL) and the language (/MODIFY-MSG-ATTRIBUTES TASK-LANGUAGE) for message output.

13.1 Message weight (severity)

EDT's messages are identified by means of a 7-digit message key. This possesses the following structure:

EDTwnnn

w Message weight

nnn Sequential number

The message weights have the following meanings:

Message weight	Meaning
0	Information
1	Warning
2	Minor error (e.g. work file too long, line length exceeded)
3	Syntax error
4 and 5	Function error (including system errors, DMS problems)
8	Error causing termination and abort

The identified message weights influence the EDT command return code (see chapter 4 "Using EDT" on page 87). Within a message weight class, the messages are grouped together by area of application on the one hand and, on the other, are sequentially numbered.

13.2 Error switch

In L mode, there are two error switches, the EDT error switch and the DMS error switch. The EDT error switch is set for the majority of messages. It indicates incorrect EDT statements. The DMS switch is set for messages that are output on file or system access errors. For some messages, both the EDT and DMS error switches are set. There are also messages for which no switch is set (e.g. information, confirmation queries, messages which are only output in F mode). The error switches can be queried in EDT procedures (see the @IF (format 1) and @RESET statements in chapter 9). Whether or not an error switch is set is indicated in the message help texts

13.3 Messages which require a response

In interactive mode, a large number of messages require the user to enter a response. The format of these messages is

```
EDTOnnn <Question>? REPLY (Y=YES; N=NO)?
```

The alternatives available here are *Yes* and *No*. The responses `Y` and `y` are interpreted as *Yes* and the responses `N` and a blank input as *No*. If the response consists of more than one character then the remaining characters are ignored. The response to characters other than those listed above depends on the operating mode. In L mode all other characters cause the question to be repeated. If no correct input has been received after the tenth repetition then the response *No* is assumed. In F mode, the entry of other characters is interpreted as *No*.

13.4 Message output

In the L mode interactive mode, messages are output to SYSOUT,and in batch mode they are output to SYSLST.

In F mode, the messages are output in the last line of the data window which is referred to as the message line. The display in the data window is then shortened by one line. If a message is longer than the message line, for example because it contains a very long *insert*, then it is output in the two last lines of the data window and the display in the data window is shortened by a further line. If the screen is split and only one data line is available then the message is truncated and output in this line. Some messages require input from the user. These are also usually output in the message line. However, in some cases, the query message is output together with another message. In such cases, this message is output in the message line and the query message is output in the screen's statement line. In all cases, the response must be entered in the first column of the statement line. If an EDT statement results in more than one message then only the message with highest weight is output. If the message weights are the same then the last generated message is output.

13.5 Message texts

EDTCOPY Copyright (C) (&00) (&01) All Rights Reserved

EDTLOAD Program '(&00)', Version '(&01)' of '(&02)' loaded from file '(&03)'

EDTSTRT Procedure '(&00)', Version '(&01)' of '(&02)' started from file '(&03)'

EDT0001 (&00) STARTED

Meaning
EDT was started.

EDT0002 (&00) RESTARTED IN COMPATIBILITY MODE

Meaning
A statement changing the operation mode was given. EDT was restarted in compatibility mode.

EDT0003 (&00) RESTARTED IN UNICODE MODE

Meaning
A statement changing the operation mode was given. EDT was restarted in Unicode mode.

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 statement has not been processed. But its syntax can only be tested at run time.
Possible reasons: indirect operands, operands in variables or user statements.
Error switch: not set.

EDT0120 TESTMODE: CHARACTER(S) SKIPPED

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

Meaning
The old content of the library element was replaced.

EDT0171 FILE '(&00)' REPLACED AND WRITTEN

Meaning
The old content of the file was replaced.

EDT0172 MEMBER '(&00)' IN LIBRARY '(&01)' CREATED AND WRITTEN

Meaning
The library element was created and written.

EDT0173 FILE '(&00)' CREATED AND WRITTEN

Meaning
The file was created and written.

EDT0178 FILE '(&00)' CLOSED

EDT0190 WORK FILE (&00) EMPTY

EDT0193 WORK FILE (&00) CLEARED

EDT0196 UFS FILE '(&00)' REPLACED AND WRITTEN

Meaning
The old content of the POSIX file was replaced.

EDT0197 UFS FILE '(&00)' CREATED AND WRITTEN

Meaning
The POSIX file was created and written.

EDT0200 CCS CHANGED TO '(&00)'

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)'

Meaning
The SDF-P list variable (&00) has been extended by appending or prefixing one or more elements to the list.

EDT0211 /FREE-VARIABLE COMMAND PROCESSED FOR S-VARIABLE '(&00)'

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.

EDT0227 ISAM FILE '(&00)' CREATED AND OPENED IN WORK FILE (&01)

Meaning
The ISAM file was created and opened in actual work file.

EDT0228 ISAM FILE '(&00)' REPLACED AND OPENED IN WORK FILE (&01)

Meaning
The old content of the ISAM file was deleted, then the file was opened in actual work file.

EDT0229 ISAM FILE '(&00)' OPENED IN WORK FILE (&01)

Meaning
The ISAM file was opened in actual work file.

EDT0230 FILE '(&00)' OPENED IN CURRENT WORK FILE (&01)

Meaning
The file was opened in actual work file.

EDT0231 FILE '(&00)' CREATED AND OPENED IN CURRENT WORK FILE (&01)

Meaning
The file was created and opened in actual work file.

EDT0232 FILE '(&00)' REPLACED AND OPENED IN WORK FILE (&01)

Meaning
The old content of the file was deleted, then the file was opened in actual work file.

EDT0235 FILE '(&00)' WRITTEN AND CLOSED

Meaning
The file was written and closed.

EDT0236 FILE '(&00)' CLOSED UNCHANGED

Meaning
The file was closed unchanged.

EDT0237 UFS FILE '(&00)' OPENED

Meaning
The POSIX file was opened in actual work file.

EDT0238 UFS FILE '(&00)' CREATED AND OPENED

Meaning
The POSIX file was created and opened in actual work file.

EDT0239 UFS FILE '(&00)' REPLACED AND OPENED

Meaning
The old content of the POSIX file was deleted, then the file was opened in actual work file.

EDT0240 UFS FILE '(&00)' CLOSED

Meaning
The POSIX file was written and closed.

EDT0241 UFS FILE '(&00)' CLOSED UNCHANGED

Meaning
The POSIX file was closed unchanged.

EDT0242 FILE '(&00)' COPIED

Meaning
The file was copied into the actual work file.

EDT0243 UFS FILE '(&00)' COPIED

Meaning
The POSIX file was copied into the actual work file.

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

Meaning
This query is issued following a @OPEN, @WRITE, @XOPEN or @XWRITE statement if the POSIX file is read only and the current user id is TSOS.

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

Meaning
The library element was opened in actual work file.

EDT0259 MEMBER '(&00)' IN LIBRARY '(&01)' CREATED AND OPENED

Meaning
The library element was created and opened in actual work file.

EDT0264 MEMBER '(&00)' IN LIBRARY '(&01)' WRITTEN AND CLOSED

Meaning
The library element was written and closed.

EDT0265 MEMBER '(&00)' IN LIBRARY '(&01)' CLOSED UNCHANGED

Meaning
The library element was 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

Meaning
The library element was opened in actual work file, the old content was not read.

EDT0274 MEMBER '(&00)' IN LIBRARY '(&01)' COPIED

Meaning
The library element was copied into the actual work file.

EDT0281 /DELETE-FILE COMMAND PROCESSED FOR FILE '(&00)'

Meaning
The file has been erased from catalog.

EDT0282 DELETE PROCESSED FOR MEMBER '(&00)'

Meaning
The element has been deleted from library.

EDT0283 UFS FILE '(&00)' DELETED

Meaning
The POSIX file has been removed from its directory.

EDT0285 SDF: SYNTAX TESTED. (&00) ERROR(S) IN RANGE

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.

EDT0293 FILE NOT WRITTEN

Meaning
N has been specified in response to an OVERWRITE inquiry.

EDT0294 MAXIMUM LINE NUMBER

Meaning
The screen cannot be completely filled with empty lines after the last used line, as not enough line numbers are available.

EDT0295 OLD COPY BUFFER CLEARED, NEW COPY BUFFER FILLED

Meaning
An R statement code is followed by a C or M statement code.
The copy buffer created by means of R statement code(s) has been cleared.

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.

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.

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.

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 for the task
DATE	- date (YYYY-MM-DD)
TIME	- time (HH:MM:SS)
STATEMENT SYMBOL	- actual statement symbol
TERMINAL	- type of terminal.

EDT0610 BUFFER SIZE UNCHANGED

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

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)' CANNOT BE SET. STANDARD WILL BE USED

Meaning
At EDT initialisation, the character set (&00) shall be made the actual character set. As this is not possible with the current operation mode, the 7 bit standard EDF03IRV is used instead.

EDT0900 EDITED FILE(S) NOT SAVED!

Meaning
A @HALT (or another) statement has been entered in order to terminate EDT, but some data have not yet been saved.
EDT will output a list of work files whose data have not yet been saved.

EDT0901 NO MATCH IN RANGE

Meaning
No match exists for the search string in the specified range when processing an @ON statement.
Error switch: EDT.
If this error occurs while an EDT procedure (@DO or @INPUT) is being processed, and logging by means of the PRINT operand has not been activated, this message is not displayed and the EDT error switch is not set.

EDT0902 FILE (&00) VERSION (&01)

Meaning
Possible reasons:
- The file (&00) has been specified in a write access statement with the version number '*' or the actual version number:
  The new actual version number after writing is (&01).
- The file (&00) has been specified in a read access statement with the version number '*' or a wrong version number:
  The correct version number is (&01).

EDT0903 FILE '(&00)' IS IN THE CATALOG, FCBTYPE = (&01)

Meaning
The file (&00) ist already in the catalog, its access method is (&01).

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

Meaning
Inquiry whether EDT is to be terminated.

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 WORK FILES USED

Meaning
A @DROP ALL statement has been issued by the user but no work files are currently used.

EDT0909 AUTOSAVE ABORTED. ERASE SAVING FILES? REPLY (Y=YES; N=NO)

Meaning
The writing of the backup files could not be performed.
Possible reasons are a virtual address space shortage or an unexpected DMS error.
The automatic saving is switched off.

Response
Y: existing saving files will be erased.
N: existing saving files will not be erased.

EDT0910 '@RENUMBER': LINES WILL BE LOST

Meaning
A @RENUMBER statement has been entered in order to renumber the lines.
If EDT renumbers 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)

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.

EDT0912 INTERRUPTION NOT POSSIBLE

Meaning
The K2-key has been pushed during a non-interruptable procedure. It is not possible to change to system mode at the moment.

EDT0913 /INFORM-PROG TO BE SIMULATED? REPLY (Y=YES; N=NO)

Meaning
The command /INFORM-PROGRAM cannot be performed in a non-interruptable procedure. EDT asks if an action should be performed.

Response
Y: EDT treats like /INFORM-PROGRAM has been given.
N: The program will be continued.

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.

EDT0915 CONVERT TO FILE CCS (&00)? REPLY (Y=YES; N=NO)

Meaning
The work file's Coded Character Set is different from that of the file to be written to.

Response
Y: convert to file's character set before writing.
N: use work file's character set for writing.

EDT0990 (&00)

Meaning
(&00): Message from test routine.

EDT0999 (&00)

Meaning
(&00): Message from external routine.

EDT1137 SPECIFIED WORK FILE IGNORED IN CONJUNCTION WITH 'SPLIT'

Meaning
In the @PAR statement a work file 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

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.

EDT1181 CODE OPERAND IGNORED

Meaning
A POSIX file, a not existing DMS file or a not existing library element shall be written using @WRITE CODE=*FILE.
For POSIX files the code set defined with @PAR CODE is used, for other files the code set of actual work file.

EDT1190 WORK FILE (&00) IS EMPTY. COPY OPERATION NOT PERFORMED

EDT1226 SPECIFIED FCBTYPE 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 FCBTYPE (&00) is taken over from the catalog.

EDT1227 CCS ATTRIBUTE CANNOT BE SET

Meaning
The file has been created or updated, but the CCS attribute cannot be set, as the access to file catalog or library catalog returned an error.
Error switch: EDT, DMS.

EDT1243 FILE '(&00)' TO BE COPIED IS EMPTY

Meaning
The DMS file to be copied into the actual work file is empty.
Error switch: EDT.

EDT1244 FILE '(&00)' EMPTY

EDT1245 JOB VARIABLE IS EMPTY

Meaning
An attempt has been made to get the value of a job variable by a @GETJV statement, but the entire job variable is empty.
Error switch: EDT.

EDT1246 UFS FILE '(&00)' TO BE COPIED IS EMPTY

Meaning
The POSIX file to be copied into the actual work file is empty.
Error switch: EDT.

EDT1247 MEMBER '(&00)' IN LIBRARY '(&01)' TO BE COPIED IS EMPTY

Meaning
The library element to be copied into the actual work file is empty.
Error switch: EDT.

EDT1248 EMPHASIS INCOMPLETE IN SOME LINES

Meaning
When the lines containing the search string are displayed on the screen after an @ON statement, not all matches are emphasized, as the insertion of screen control characters makes one or more lines too long.
Error switch: EDT.

EDT1253 (SOME) RECORD(S) TRUNCATED

Meaning
Records that are too long are truncated when being read into the internal work file (in Unicode Mode after 32768, otherwise after 256 characters).
Error switch: EDT.

EDT1254 NO MARKS SET FOR FILE TO BE PROCESSED IN REAL MODE

Meaning
No marks can be set for files processed in real mode (statement @OPEN).
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
In Unicode mode, a line exceeds the maximum length (32768 characters) during tabulator expansion. The line will be truncated.
In compatibility mode, when reading data in line mode or the element of a list variable, more than 256 characters are read. The input will be truncated.
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 is not loaded.
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.

EDT1937 MODIFIED LINE > 32768 CHARACTERS

Meaning
A work file line would become too long as a result of modification with an @ON statement. The line will remain unchanged.
Error switch: EDT.

EDT1938 MODIFIED LINE > 32768 CHARACTERS

Meaning
A @DO procedure line would become too long as a result of parameter replacement. The line will be truncated.
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 input record in F mode is longer than the LIMIT specified in the @PAR statement, it is truncated.
(&00): maximum permissible record length.

EDT2301 COPY BUFFER OVERFLOW

Meaning
The copy buffer cannot hold more than 256 line numbers.

EDT2400 LINE TRUNCATED AFTER 32768 CHARACTERS

Meaning
A @CREATE or @SETJV statement, a J statement code or a data input with tabulator expansion results in a work file line with more than 32768 characters. The line will be truncated after 32768 charcters.
Error switch: EDT.

EDT2401 LINE TRUNCATED AFTER 2048 BYTES

Meaning
The prompt string to be displayed for @CREATE statement (format 3 or 4) is too long. It is truncated after 2048 bytes.
Error switch: EDT.

EDT2402 OUTPUT TRUNCATED AFTER (&00) CHARACTERS

Meaning
The prompt string to be output for statement @CREATE (format 3 or 4) is too long. It is truncated after (&00) characters.
Error switch: EDT.

Response
Shorten string.

EDT2403 OUTPUT TRUNCATED AFTER 4096 BYTES

Meaning
The string to be assigned to a S variable is too long.
It is truncated after 4096 bytes.
Error switch: EDT.

EDT2404 NOT ENOUGH LINES FOR HEX MODE

Meaning
On a split screen there are less lines in the work window than would be needed for diplaying a line in hexadecimal mode.
Error switch: EDT.

Response
Enlarge the work window (@PAR SPLIT).

EDT2405 LINE TRUNCATED DURING AUTOSAVE

Meaning
Whe a work file is saved using @AUTOSAVE, a line is longer than the maximum record line of the saving file. The line will be truncated.
Error switch: EDT.

EDT2406 MAXIMUM LINE NUMBER

Meaning
The given statement was processed correctly, but the new actual line can not be the last line of the work file incremented with the actual increment, because the maximum line number (9999.9999) would be exceeded. The new actual line will become the last line of the work file.
Error switch: EDT.

EDT2407 OUTPUT OF SYSTEM COMMAND TRUNCATED

Meaning
The output of the operating system command issued via @SYSTEM is too long, it will be truncated.
Error switch: EDT.

EDT2408 LOGGING TERMINATED

Meaning
The output media specified in @LOG cannot be written to. Logging will be switched off.
Error switch: EDT.

EDT2409 CPU TIME SPECIFIED AT EDT START EXCEEDED

Meaning
The CPU time specified at EDT start has been exceeded. EDT will be continued.
Error switch: EDT.

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 line number in the work file is set to 0.0001 for this line.
Error switch: EDT.

EDT2901 CHECK LINE LENGTH

Meaning
The line length check is active (@CHECK or @TABS) and the line read or created by tabulator expansion is longer than specified in the CHECK statement or operand. The line will be created as specified.
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
Possible error cause:
- An empty file on disk is accessed by means of a @READ, @GET, @INPUT or @ELIM statement.
- A @SETLIST statement has been specified in an empty work file.
- A work 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
The statement contains a syntax error.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3003 '(' MISSING

Meaning
Error switch: EDT.

Response
Insert the missing bracket and re-enter the statement.

EDT3004 ')' MISSING

Meaning
Error switch: EDT.

Response
Insert the missing bracket and re-enter the statement.

EDT3040 NAME INVALID OR MISSING

Meaning
The string contains more than 8 characters or does not fullfill the syntax of the operand or is missing.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3050 INVALID SYSLST-NUMMER

Meaning
The SYSLST number issued in the statement @LOG is invalid.
Only values between 1 and 99 are valid.
Error switch: EDT.

Response
Correct and re-enter the statement.

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.

Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3066 WRONG COLUMN NUMBER

Meaning
A wrong column number has been specified in the @SETF statement.
Error switch: EDT.

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
The operand specification in the @PAR statement contains an error, ON, OFF or O is missing after an equals sign.
Error switch: EDT.

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.
Error switch: EDT.

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 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.
Error switch: EDT.

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.
Error switch: EDT.

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 character ':' is missing in the specified column range, or the specified range is invalid.
Error switch: EDT.

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.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3082 LINE NUMBER 0 INVALID

Meaning
Line number 0 or increment 0 cannot be specified.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3085 '(&00)' NOT POSSIBLE FOR PLAM ELEMENT TYPE '(&01)'

Meaning
Library elements of the type R, C, H, L, U, F or corresponding free type cannot be used with the statements @COPY, @OPEN, @WRITE or @INPUT.
Error switch: EDT.

Response
Correct and re-enter the statement.

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.
Error switch: EDT.

Response
Correct and re-enter statement.

EDT3088 INVALID NAME OF S-VARIABLE

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

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.

Response
Correct and re-enter the statement.

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.

Response
Correct and re-enter the statement.

EDT3100 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.

EDT3101 INVALID STATEMENT

Meaning
No valid EDT statement has been recognized.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3106 SPECIFIED WORK FILE INVALID (VALID RANGE: 0..9)

Response
Correct and re-enter the statement.

EDT3110 EQUAL SIGN EXPECTED. STATEMENT NOT PROCESSED

Meaning
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3111 'TYPE' OPERAND IS MISSING OR INVALID

Meaning
The TYPE operand has been specified without a valid operand value.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3112 'TYPE' OPERAND ALREADY DEFINED

Meaning
An attempt was made to specify the TYPE operand a second time.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3116 'CODE' OPERAND MISSING OR INVALID

Meaning
The CODE operand has been specified without a valid operand value.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3117 'MODE' OPERAND MISSING OR INVALID

Meaning
The MODE operand has been specified without a valid operand value.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3118 'MODE' OPERAND ALREADY DEFINED

Meaning
An attempt was made to specify the MODE operand a second time.
Error switch: EDT.

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

Meaning
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3122 FILE NAME MISSING OR FORMAT OF SPECIFIED FILE NAME INVALID

Meaning
Error switch: EDT.

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.
Error switch: EDT.

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.
Error switch: EDT.

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
No name has been specified for the work file, or the specified name is invalid.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3128 PLAM LIBRARY NAME INVALID OR MISSING

Meaning
Error switch: EDT.

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 FCBTYPE=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 of the PLAM member has not been specified in the statement, or the specification is invalid.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3133 PLAM VERSION SPECIFICATION INVALID

Meaning
Error switch: EDT.

Response
Correct the version specification 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
More than one character has been specified as a symbol.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3170 SYNTAX ERROR IN LINE NUMBER

Meaning
The operand which is supposed to be a line number is syntactically incorrect.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3172 MODULE NAME TOO LONG

Meaning
The modul name specified in an @UNLOAD statement was longer than 8 characters (compatible format) or longer than 32 characters (new format).
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3173 NUMBER OF WORK FILE FOR COMPARE OPERATION MISSING OR INVALID

Meaning
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3174 NAME TOO LONG

Meaning
A string used for specifying a file or jobvariable name consists of more characters than allowed by the system (fully qualified: 54, with wildcards: 80).
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3175 SYNTAX ERROR IN SPECIFIED RANGE

Meaning
Error switch: EDT.

Response
Correct and re-enter the statement.

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 (compatible format) or 32 characters (new format).
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.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3179 ENTRY NAME MISSING OR INVALID

Meaning
The specified string does not fulfill the syntactical requirements for entry names, or the entry name is missing.
Error switch: EDT.

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 chartacter 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 the statement @SHOW, @FSTAT, @STATUS, @SYSTEM or @STAJV.
Error switch: EDT.

Response
Correct and re-enter statement.

EDT3190 SPECIFIED NUMBER INVALID (VALID RANGE: 1..32768)

Meaning
The specified value is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3191 OPERAND TOO LONG

Meaning
The specified operand is too long.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3192 ILLEGAL CHARACTER IN OPERAND

Meaning
The specified operand contains illegal characters.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3193 SPECIFIED WORK FILE INVALID (VALID RANGE: 0..22)

Meaning
The specified work file number is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3194 SPECIFIED NUMBER INVALID (VALID RANGE: 1..9)

Meaning
The specified record mark is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3195 SPECIFIED UNICODE VALUE INVALID

Meaning
The specification of a separator character, a structure symbol, a surrogate character or a filler character as a Unicode value does not correspond to a valid character.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3196 SPECIFIED HEX OR BINARY VALUE INVALID

Meaning
The specified haxadecimal or binary value does not correspond to a valid character.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3197 SPECIFIED NUMBER INVALID (VALID RANGE: 0..32768)

Meaning
The specified value is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3198 SPECIFIED NUMBER INVALID (VALID RANGE: 0..99999999)

Meaning
The specified value is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3199 SPECIFIED NUMBER INVALID (VALID RANGE: 0..31)

Meaning
The specified value is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3200 SPECIFIED NUMBER INVALID (VALID RANGE: 1..8)

Meaning
The specified value is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3201 SPECIFIED NUMBER INVALID (VALID RANGE: 0..65535)

Meaning
The specified value is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3202 SPECIFIED NUMBER INVALID (VALID RANGE: 1..65535)

Meaning
The specified value is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3203 'KEY' OPERAND ALREADY DEFINED

Meaning
The KEY operand has been specified a second time.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3204 'CODE' OPERAND ALREADY DEFINED

Meaning
The CODE operand has been specified a second time.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3205 'KEY' OPERAND IS MISSING OR INVALID

Meaning
The KEY operand has been specified without a valid operand value.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3206 SPECIFIED NUMBER INVALID (VALID RANGE: 0..255)

Meaning
The specified value is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3207 SPECIFIED NUMBER INVALID (VALID RANGE: 0..256)

Meaning
The specified value is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3209 SPECIFIED WORK FILE INVALID (VALID RANGE: 1..22)

Meaning
The specified work file number is outside the valid range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3210 'LENGTH' OPERAND IS MISSING OR INVALID

Meaning
The LENGTH operand has been specified without a valid operand value.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3211 'LENGTH' OPERAND ALREADY DEFINED

Meaning
The LENGTH operand has been specified a second time.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3212 'MARK' OPERAND IS MISSING OR INVALID

Meaning
The MARK operand has been specified without a valid operand value.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3213 'MARK' OPERAND ALREADY DEFINED

Meaning
The MARK operand has been specified a second time.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3214 LINK NAME IS MISSING OR INVALID

Meaning
The specified link name has no valid value.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3215 MISSING CLOSING QUOTE IN STRING

Meaning
A string was specified without a closing quote symbol.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3216 INVALID RANGE (LOWER > UPPER)

Meaning
The first switch number in the @SETSW statement is higher than the second one.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3217 TOO MANY DIGITS IN NUMBER

Meaning
The decimal integer used for numbering lines in the @SEQUENCE statement may only have up to 8 digits.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3218 'TO' OPERAND EXPECTED

Meaning
In the @COPY or @MOVE statement a TO operand is missing.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3219 'TYPE' OPERAND NOT ALLOWED

Meaning
The TYPE operand is only allowed for DMS files.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3220 'KEY' OPERAND NOT ALLOWED

Meaning
The KEY operand is only allowed for DMS files.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3221 'CODE' OPERAND NOT ALLOWED

Meaning
The CODE operand is only allowed for POSIX files.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3222 INCREMENT MISSING OR INVALID

Meaning
No increment is specified or the increment value is invalid.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3223 UFS FILE NAME MISSING OR INVALID

Meaning
The specification of a UFS file name is missing, or it does not follow the syntax of a file name in the POSIX file system.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3224 SPECIFIED NUMBER INVALID (VALID RANGE: 1..2048)

Meaning
The factor n in the repetition statement # is not within the permissible range.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3901 ILLEGAL BINARY CONSTANT

Meaning
A string with a ’B’ in front of the first single quote or the text input after @INPUT BINARY is in error.
Only the digits ’0’ and ’1’ are valid characters, and the string must not be empty.
Error switch: EDT.

Response
Correct and re-enter the statement or text.

EDT3902 ILLEGAL HEX CONSTANT

Meaning
A string with an ’X’ in front of the first single quote or the text input after @INPUT HEX is in error. 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.

Response
Correct and re-enter the statement or text.

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.

Response
Correct and re-enter the statement.

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.

Response
Correct and re-enter the statement.

EDT3906 LINE NUMBER INVALID

Meaning
Possible error cause:
- The integer variable in the statement @SET (format 3) does not contain a valid line number.
- The target line number variable in the statement @SET (format 4 or 5) does not contain a valid line number.
- The destination of the output in the statement @GETJV is not a valid line number.
- The line number is too high for the specified KEYLEN of a file opened in real mode by means of an @OPEN statement.

Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3907 EMPTY STRING NOT PERMISSIBLE

Meaning
A string specified directly or indirectly (i.e. 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.

Response
Correct and re-enter the statement.

EDT3908 STRING MISSING OR INVALID

Meaning
A string is missing in a statement or is invalid.
In compatibility mode, the error cause also may be that no file name is specified in an I/O statement and neither a local nor a global @FILE entry is defined.
Error switch: EDT.

Response
Correct and re-enter the statement.

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

Meaning
A formal operand has been specified at least twice in the @PARAMS statement.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3911 DUPLICATE KEYWORD

Meaning
A keyword has been specified at least twice in a @DO statement.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3922 INVALID COLUMN (RANGE)

Meaning
The value specified for a column is invalid, or the specified column (range) is syntactically incorrect.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3951 PROCEDURE NUMBER > 22

Meaning
Error switch: EDT.

EDT3952 INVALID SYMBOL

Meaning
For the symbol definition a special character has to be choosen.
Error switch: EDT.

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 without any additional information.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT3999 (&00)

Meaning
The external routine reports a syntax error in the specified statement with the message (&00).
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT4200 MACRO '(&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 MACRO '(&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 MACRO '(&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 occured.
(&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 MACRO '(&00)': XHCS ERROR CODE: '(&01)'

Meaning
All XHCS errors are output in this form:
(&00): XHCS macro (NLSCODE, etc.) during whose processing the error occured
(&01): Hexadecimal error code.
or more detailed information about the XHCS error
see the BS2000 manual "XHCS".
Processing of an @INPUT file is aborted due to this error.
Error switch: DMS.

EDT4204 MACRO '(&00)': TIAM ERROR CODE: '(&01)'

Meaning
All TIAM errors are output in this form:
(&00): TIAM-Macro (WRLST, etc.) during whose processing the error occured
(&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 MACRO '(&00)': BLS ERROR CODE: '(&01)'

Meaning
All errors of the dynamic binder loader are output in this form:
(&00): BLS macro (BIND) during whose processing the error occured
(&01): Hexadecimal error code.
For more detailed information about the BLS error see the BS2000
manual "Dynamic Binder Loader" or the BS2000 manual "Macro cals".
Processing of an @INPUT file is aborted due to this error.
Error switch: DMS.

EDT4206 POSIX-CALL '(&00)': ERROR '(&01)'

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 MACRO '(&00)': SDF ERROR CODE: '(&01)'

Meaning
All errors of SDF-macros are output in this form:
(&00): SDF-Macro (CMDSTA, etc.) during whose processing the error occured
(&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.

EDT4208 MACRO '(&00)' RETURNS ERROR CODE '(&01)'

Meaning
When executing the macro (&00) an error with error code (&01) occurred.
The error code is issued in form of a message number if available.
Otherwise the complete macro returncode (subcode2, subcode1, maincode) is issued. If there are two return codes available, both are issued, separated by a /.
For more detailed information about the error see the appropriate BS2000 manual.
Error switch: EDT, DMS.

EDT4209 FUNCTION '(&00)' RETURNS ERROR '(&01)'

Meaning
When executing the function (&00) an (&01) error was returned in C-variable ERRNO.
For more detailed information about the error see the appropriate BS2000 manual.
Error switch: EDT, DMS.

EDT4300 ERROR AT SYSTEM COMMAND: ERROR CODE '(&00)'

Meaning
The command specified in the @SYSTEM statement is rejected by the CMD macro.
For more detailed information about the error cause enter the command /HELP-MESS (&00) in system mode, or see the BS2000 manual "System Messages".
Error switch: DMS.

EDT4310 SDF: SYNTAX ERROR IN LINE (&00)

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.

Response
Enable SDF guided error dialog, e.g. by
@SYSTEM '/MODIFY-SDF-OPTIONS GUIDANCE=MIN'.

EDT4312 CHECK TAB COLUMNS IN LINE (&00)

Meaning
The CHECK function was specified for the @TABS statement. CHECK detected that in line (&00) tabs caused reverse positioning, i.e. text was overwritten.
Processing of @TABS RANGE has been aborted.
Error switch: EDT.

EDT4313 LINE (&00) > 256 CHARACTERS

Meaning
The length of line (&00) would exceed 256 characters.
Processing of @TABS RANGE has been aborted.
Error switch: EDT.

EDT4314 LINE (&00) > 32768 CHARACTERS

Meaning
The length of line (&00) would exceed 32768 characters.
Processing of @TABS RANGE has been aborted.
Error switch: EDT.

EDT4315 RANGE SYMBOL MUST NOT BE USED AS STATEMENT SYMBOL

Meaning
For the definition of a new statement symbol with statement @:
the actual range symbol (see @RANGE) was used.
The statement is not processed.
Error switch: EDT.

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
The @INPUT statement is not permissible inside an @INPUT procedure.
Error switch: EDT.

EDT4903 BOTH OPERANDS IN '@QUOTE' STATEMENT ARE THE SAME

Meaning
If both operands 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 WORK FILE

Meaning
The statement (&00) refers to the current work file and therefore is impossible to be executed (e.g. @DO, @DROP).
Error switch: EDT.

EDT4907 '@DROP' NOT POSSIBLE DURING PROCEDURE FILE PROCESSING

Meaning
The @DROP statement is not permissible inside a @DO procedure.
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 @SYSTEM.

EDT4909 WORK FILE ALREADY ACTIVE

Meaning
In a @PROC statement the number of current work file was specified.
Error switch: EDT.

EDT4910 S-VARIABLE MUST BE OF TYPE LIST

Meaning
The SDF-P variable specified in a @GETLIST or @SETLIST statement or in a @LOG statement in compatibility mode is not of type LIST
or has not been declared yet.
Error switch: EDT.

Response
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.

EDT4916 FILE '(&OO)' 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 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, @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
The specified statement is not permissible inside a @DO or @INPUT procedure (e.g. @INPUT, @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.
In compatibility mode, a possible error cause is that the file name specified in single quotes or in a string variable has a leading blank.
Error switch: EDT, DMS.

EDT4924 INVALID FORMAL OPERAND

Meaning
A @PARAMS statement contains an invalid formal operand.
Error switch: EDT.

EDT4925 STATEMENT ONLY PERMITTED IN WORK FILE 0

Meaning
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:
"EDT8001 EDT 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
Possible error cause:
- In a @COMPARE statement (format 1) the value of int2 is greater than that of int1.
- The string specified after STRING keyword in a @SET statement (format 3) cannot be interpreted as a line number.
- A @PARAMS keyword or a @DO operand has an invalid value.

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(REC-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, or the range of lines specified in the @COMPARE statement is invalid.
Error switch: EDT.
If a non existing line number has been specified for a string while an EDT procedure (@DO or @INPUT) is being processed, and logging has not been activated by means of the PRINT operand, this message is not displayed and the EDT error switch is not set either. However the statement will not be processed.

EDT4933 MODULE LOADING NOT POSSIBLE

Meaning
It is not possible to load the module (e.g. EDTSTRT) by means of the @RUN or @USE statement.
Error switch: EDT.

EDT4934 FILE HAS ACCESS METHOD 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 FILE IS OPENED REAL

Meaning
An attempt was made to process a statement that is not permissible as long as the file is opened real by means of @OPEN (e.g. @RENUMBER).
Error switch: EDT.

EDT4936 'KEY-POSITION <>5' AND 'RECORD-FORMAT=*VARIABLE' NOT SUPPORTED

Meaning
It is not possible to process a file with that catalog properties by means of @GET, @SAVE or @OPEN.
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 and the software tabulators (in Unicode mode) must be in ascending order.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT4941 NO POSITIONS DEFINED

Meaning
Tabulators cannot be activated 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 or @GOTO statement (in compatibility mode also an @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

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.

Response
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
Possible error cause:
- An attempt was made to change the screen dimension with a @VDT statement for a terminal other than a 9763.
- An attempt was made to define hardware tabulators for a 3270 terminal.

Error switch: EDT.

EDT4946 OVERFLOW ERROR

Meaning
The result of an integer expression 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.
Error switch: EDT.

EDT4950 WORK FILE IS UNDEFINED

Meaning
No work file was specified in the @DO statement and no work file has been left with @END before.
In compatibility mode, a possible error cause also may be that in a @DO or @COMPARE statement a procedure file has been specified that has not been used yet.
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.
Error switch: EDT.

EDT4952 NO SEPARATOR DEFINED

Meaning
The statement @SEPARATE has been issued without operator AT. As there has not been defined a separator with @PAR SEPARATOR before, the statement cannot be performed.
Error switch: EDT.

Response
Issue the character for separation in the statement @SEPARATOR after the operand AT or preset it with @PAR SEPARATOR.

EDT4953 NO CHARACTER FOR TABULATOR DEFINED

Meaning
The statement @TABS RANGE has been issued. But there has not been activated a character for tabulator yet.
Error switch: EDT.

Response
Define and activate a character and positions for the tabulator before repeating @TABS RANGE.

EDT4954 REQM ERROR. PLEASE ACKNOWLEDGE. REPLY (Y=YES)

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 there are nested procedure files which are not yet 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 "EDT8001 EDT 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.
Error switch: DMS.

EDT4959 WORK FILE ALREADY ACTIVE

Meaning
A work file that is currently used to execute a @DO procedure (active work file), can not be made the actual work file.
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 "EDT8001 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
Only 22 procedure files can be processed at the same time.
The @DO statement is rejected.
Error switch: EDT.

EDT4962 TOO MANY NESTED PROCEDURE FILES

Meaning
Only 22 procedure files can be nested. The @PROC statement is rejected.
In compatibility mode, the @INPUT statement is rejected too.
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 LINE NUMBER AREA EMPTY

Meaning
A @ statement (or in Unicode mode a @SET statement, format 6) has been specified to make the last stored pair of line number and increment the actual line number and increment. However no more entries exist in the line number area.
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 UPDATE operand has been specified for an ISAM file.
Error switch: EDT.

EDT4967 'UPDATE' FOR SAM FILE NOT POSSIBLE

Meaning
A @SAVE statement with UPDATE operand has been specified for a SAM file.
Error switch: EDT.

EDT4968 WORK FILE NOT EMPTY

Meaning
There are still some lines in the work file. Opening a file for real processing using @OPEN statement 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
The @ELIM statement is only allowed for ISAM files.
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 at the moment.
Possible reasons are:
- running under an userid with special privileges
- running in a non-interruptable procedure
- statement has been inhibited by the calling program.

Error switch: EDT.

EDT4977 'RECORD-FORMAT=*UNDEFINED' ILLEGAL

EDT4978 INVALID IN F-MODE

Meaning
The statement is not allowed in full screen 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) or the CCSN
specified in the CODE operand of a Unicode statement is unknown.
Error switch: EDT.

EDT4981 RECORD-SIZE > 256. FILE NOT WRITTEN

EDT4982 REQUESTED JOB VARIABLE NOT CATALOGED

Meaning
The name of the job variable specified in a @GETJV, @STAJV, or @ERAJV statement has not been found in the catalog, or no name was specified in a @STAJV statement and no job variables exist in the actual user id.
Error switch: EDT, DMS.

EDT4983 CHANGE OF OPERATION MODE NOT POSSIBLE

Meaning
A statement was given to switch from compatibility mode to Unicode mode (@MODE, @CODENAME) or vice versa (@MODE). However a mode change is only possible if all work files are empty and no files are opened.
If the statement for switching to compatibility mode was read from an EDT start procedure or from SYSDTA, the character set of this input file additionally must be useable in compatibility mode.
Error switch: EDT.

Response
Close opened files (@CLOSE), delete work files (@DELETE) and re-enter the statement. If necessary, correct input file.

EDT4984 NON-NUMERIC KEY

Meaning
When reading a file an attempt was made to access a record whose key cannot be made a EDT line number. Processing of the statement was aborted.
Error switch: EDT, DMS.

EDT4985 WRONG VERSION FOR FILE (&00), CORRECT VERSION (&01)

Meaning
The file (&00) has been specified in a write access statement with a wrong version number. The correct version number is (&01).
The statement will not be processed.
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.

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.

EDT5122 NO FILE NAME

Meaning
No file name has been specified in the @WRITE or @XWRITE statement and no file is opened for writing back.
Error switch: EDT.

EDT5126 '(&OO)' 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.

EDT5177 NO FILE TO CLOSE

Meaning
A @CLOSE statement was given, but no file is opened.
Error switch: EDT.

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.
Error switch: EDT.

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
Neither in the statement itself nor using the @PAR LIBRARY statement a library name has been defined.
Error switch: EDT.

EDT5188 NUMBER OF LINES NOT PERMISSIBLE

Meaning
The number specified in the SPLIT statement or 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.
Error switch: EDT.

EDT5189 '(&00)' NOT POSSIBLE: A FILE IS OPENED IN WORK FILE 9

Meaning
A (&00) 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, @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
Possible reasons:
- The file specified in a @COPY, @OPEN or @WRITE statement cannot be processed by EDT because its access-method is neither SAM nor ISAM.
- An attempt was made to write to a not yet existing file associated with the link name EDTISAM and the access method SAM using the @SAVE statement.

Error switch: EDT, DMS.

Response
Convert the specified file into a SAM or ISAM file resp. change the access method.

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

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

Meaning
The ISAM file has been written. While restoring the secondary keys afterwards an error has been reported.
Error switch: EDT.

Response
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, @WRITE, @DELETE, @INPUT or @SHOW statement or predefined in a @PAR statement cannot be accessed by PLAM.
Error switch: EDT.

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 already exists.
The statement has not been processed.
Error switch: EDT.

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

Meaning
The library specified in the statement cannot be processed, as it does not exist.
Error switch: EDT.

EDT5268 MEMBER '(&00)' IS LOCKED

Meaning
The specified member could not be accessed, as it is either protected against unauthorized access or has already been opened.
Error switch: EDT.

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

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

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

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 specified member could not be copied to the virtual data space because the member does not exist.
The statement (@COPY, @INPUT Format2) has not been processed.

Response
Check PLAM typ of required member.

EDT5278 FILE '(&00)' PROTECTED BY PASSWORD

Meaning
Error switch: EDT, DMS.

Response
Contact the file owner.

EDT5279 FILE '(&00)' LOCKED

Meaning
The specified file either has already been opened or is read-protected in compatibility mode.
Error switch: EDT, DMS.

EDT5281 FILE '(&00)' DOES NOT EXIST

Meaning
The file specified in the statement cannot be processed, as it does not exist.
Error switch: EDT.

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.
Error switch: EDT.

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

Meaning
The requested library element is not editable. The specified user type is not equivalent to one of the PLAM types S,M,P,J,D or 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 of the specified type exist, or the library is empty.
Error switch: EDT.

EDT5289 LINK NAME FOR JOB VARIABLE NOT DEFINED

Meaning
An attempt was made to access a job variable 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 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 (@STAJV or @ERAJV) with a partially qualified 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.
Error switch: EDT.

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.

EDT5295 NO MORE MEMORY AVAILABLE

Meaning
No more virtual memory is available for file processing.
Error switch: EDT.

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

Meaning
The specified UFS-file (&00) cannot be dealt with, for it does not exist.
Error switch: EDT.

EDT5311 UFS FILE '(&00)' ALREADY EXISTS

Meaning
The file specified in @OPEN, @WRITE, @XOPEN oder @XWRITE cannot be handled with the operand MODE=NEW, for it already exists as UFS-file.
Error switch: EDT.

EDT5312 INVALID ACCESS TO UFS FILE '(&00)'

Meaning
The UFS file whose name was specified in an @OPEN, @COPY, @WRITE, @INPUT, @XOPEN, @XCOPY oder @XWRITE statement could not been opened for reading or writing 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

Meaning
The user issued @SDFTEST PROGRAM or marked a line starting with '//' with statement code T, but has not defined a program name yet.
Error switch: EDT.

Response
Issue @SDFTEST with operand PROGRAM=name or define a program name with @PAR SDF-PROGRAM=name.

EDT5321 SDF: PROGRAM NAME UNKNOWN

Meaning
The 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

Meaning
Processing of @SDFTEST or the statement code T has been aborted.
A possible reason is that the statement to be tested has 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.

EDT5324 SDF: SYNTAX TEST ABORTED BY USER

Meaning
The syntax check of SDF commands or SDF statements using @SDFTEST has been aborted by user.
Error switch: EDT.

EDT5325 SDF: LINE TOO LONG

Meaning
A line to be checked syntactically exceeds the maximum length allowed by SDF (16379 characters). The syntax check will be aborted.
Error switch: EDT.

EDT5326 SDF: OUTPUT TOO LONG

Meaning
The output for a line checked by SDF exceeds the maximum length allowed by SDF (16379 characters). The syntax check will be aborted.
Error switch: EDT.

EDT5327 CANNOT CONVERT TO TERMINAL CCS

Meaning
The actual character set cannot be converted into the terminal character set in order to allow a SDF error dialog. The syntax check will be aborted.
Error switch: EDT.

EDT5340 CANNOT GET S-VARIABLE

Meaning
The statement @GETVAR or @GETLIST could not be processed because the specified variable is not set to any value, or the list does not contain any element.
Error switch: EDT.

EDT5341 S-VARIABLE LONGER THAN 256 CHARACTERS

Meaning
The @GETVAR statement could not be performed, for the value of the specified variable is a string longer than 256 characters.
Error switch: EDT.

EDT5342 WRONG TYPE OF S-VARIABLE

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 equal sign.
Error switch: EDT.

Response
A SDF-P variable of type INTEGER can only be put to an integer variable and vice versa.

EDT5343 WRONG TYPE OF LIST ELEMENT

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: EDT.

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
When building data for screen output, an unrecoverable format error occurrred. EDT will be terminated.

EDT5354 STRUCTURE SYMBOL '(&00)' NOT FOUND

Meaning
The structure symbol (&00) does not exist in the specified line.
No positioning has been performed.
Error switch: EDT.

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 <ln> does not exist.
Error switch: EDT.

EDT5358 LINE SHORTER THAN 256 BYTES

Meaning
The line specified in format 1 of the @CODE statement by <ln> 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 the statement code A/B/O cannot be processed.

EDT5362 <TEXT> SPECIFICATION ILLEGAL IN CURRENT STATEMENT

EDT5364 NO INSERT: MAXIMUM LINE NUMBER

Meaning
New data lines cannot be inserted at the end of work file by means of a statement or a statement code because this would cause the maximum permissible number of lines to be exceeded.
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=OFF 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 a data display terminal without p-keys.

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.

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 USER SYMBOLS ARE PERMITTED

Meaning
5 is the maximum permissible number of user statement symbols that can be specified using a @USE statement.
Error switch: EDT.

Response
Delete a symbol by means of the @USE statement and define a new symbol.

EDT5375 NO 'USE' ENTRY DEFINED WITH SPECIFIED SYMBOL

Meaning
The specified USE entry has not been defined and thus cannot be deleted.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT5376 STATEMENT BUFFER EMPTY

Meaning
The SHIH statement has not been processed because no statements have been stored previously in the statement buffer.

EDT5380 SOME JOB VARIABLES NOT ERASED

Meaning
An attempt was made to erase all job variables whose name was specified as partially qualified or using wildcards,
but some of these job variables 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.

EDT5381 JOB VARIABLES NOT ERASED

Meaning
More than one job variable matches the string specified in the @ERAJV statement given in batch mode without an ALL operand.
The @ERAJV statement will not be processed.
Error switch: EDT.

EDT5400 NOT SUPPORTED ON THIS INTERFACE

Meaning
The statement is not possible in this work mode or at this subroutine interface.
Error switch: EDT.

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

Meaning
A statement filter only can be defined if the @USE statement is issued at the subroutine interface.
Error switch: EDT.

EDT5410 UNDEFINED ERROR IN USER PROGRAM

Meaning
EDT received an undefined return code from a user program without any additional information.
Error switch: EDT.

Response
Correct the user program.

EDT5419 (&00)

Meaning
EDT received an undefined return code from a user program with the message (&00).
Error switch: EDT.

Response
Correct the user program.

EDT5442 TERMINAL BUFFER TOO SMALL. SCREEN FORMAT SET TO F1

Meaning
With the choosen screen format, the length of screen output would exceed the terminal buffer size. The screen format is changed to F1.
Error switch: EDT.

EDT5443 TERMINAL BUFFER TOO SMALL. CHANGED TO L-MODE

Meaning
The length of screen output in F mode would exceed the terminal buffer size. The operating mode is changed to L mode.
Error switch: EDT.

EDT5444 FILE WRITE NOT POSSIBLE. RECORD LONGER THAN (&00) BYTES

Meaning
When writing a file using a @WRITE, @CLOSE or @SAVE statement, or writing in a file opened in real mode, a record would exceed the maximum file record size (&00). The record size may be limited by the blocking factor or a fixed record length specification. It should be noted that for Unicode character sets the record size in bytes may be greater than the number of characters in the work file line, and that it may depend on the chosen character set.
The file writing will be aborted or the real opened file will be closed.
Error switch: EDT.

EDT5445 DUPLICATE KEYS IN REAL OPENED FILE DETECTED

Meaning
In a file opened in real mode by means of an @OPEN statement duplicate keys were detected. Processing of the file will be aborted.
Error switch: EDT.

EDT5446 LINE NUMBER TOO BIG

Meaning
In a file opened in real mode by means of an @OPEN statement a line number cannot be used as key because it is too big for the specified key length of the file.
Error switch: EDT.

EDT5447 PRINTING FAILED

Meaning
The immediate output to the printer has not been processed, because an error occurred during the print system call.
Error switch: EDT, DMS.

EDT5448 CANNOT USE TEMPORARY FILE

Meaning
A temporary file is needed for immediately outputting a work file range to the printer. However it is not available due to the actual BS2000 system parameters.
Error switch: EDT.

EDT5449 NOT ENOUGH LINES FOR HEX MODE

Meaning
On a split screen there are less lines in the work window than would be needed for diplaying a line in hexadecimal mode.
Error switch: EDT.

Response
Enlarge the work window (@PAR SPLIT).

EDT5450 NATIONAL EDT: INTERNAL ERROR. RETURNCODE = X'(&00)' AT CCS (&01)

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.

EDT5451 CANNOT OPEN TAPE FILE

Meaning
A magnetic tape file cannot be opened by means of a @OPEN statement.
Error switch: EDT.

EDT5452 CHANGE OF CCS NOT POSSIBLE FOR REAL OPENED FILES

Meaning
An attempt was made to change the coded character set of the work file in which a file is opened in real mode, or the coded character set of a file to be opened in real mode is different from that of the actual work file.
The @CODENAME resp. @OPEN statement will not be processed.
Error switch: EDT.

EDT5453 SOME CHARACTER CANNOT BE CONVERTED

Meaning
Some characters of the string are not defined in the target character set.
The string cannot be converted.
Error switch: EDT.

Response
Define substitution character using @PAR SUBSTITUTION-CHARACTER.

EDT5454 ILLEGAL BYTE SEQUENCE IN INPUT DATA

Meaning
In a file, a job variable, a S variable or a record delivered over the subroutine interface there is a byte sequence, which does not correspond to a valid character in the target character set. The file, the variable or the record will not be read.
Error switch: EDT.

EDT5456 SOME CHARACTERS CANNOT BE CONVERTED AND SOME LINES MUST BE TRUNCATED

Meaning
A check with @CHECK (Format 2) resulted both in characters not contained in the target code set and in lines or string variables whose length is longer than the maximum length allowed.
Error switch: EDT.

Response
Define substitution character using @PAR SUBSTITUTION-CHARACTER.
Split or shorten lines or string variables.

EDT5457 FILE CCS DIFFERENT FROM WORK FILE CCS, FILE NOT WRITTEN

Meaning
The coded character set of the work file is different from the coded character set of the file the work file should be written to. The file will not be written.
Error switch: EDT.

Response
Specify operand CODE in @CLOSE or @WRITE statement.

EDT5458 CCS CANNOT BE CHANGED

Meaning
In a @CREATE statement a coded character set is specified for the actual work file. However this work file already has a different coded character set. The @CREATE statement will not be processed.
Error switch: EDT.

EDT5459 CANNOT CREATE LINE NUMBER FROM ISAM KEY

Meaning
When reading an ISAM file whose key position is different from standard, whose key length is too big or whose keys are not numeric, the line number shall be built from ISAM key. The file will not be read.
Error switch: EDT.

Response
Read the file using KEY=DATA.

EDT546O CANNOT CONVERT HEX OR BIN CHARACTER TO WORK FILE CCS

Meaning
A hexadecimal or binary value does not correspond to a valid character in the work file's coded character set. The input is rejected.
Error switch: EDT.

EDT5461 QUOTE SYMBOL MUST BE DIFFERENT TO WILDCARD SYMBOLS

Meaning
An attempt was made to define a character as string delimiter symbol (single or double quote) which is currently used as wildcard symbol (see @SYMBOLS). The @QUOTE statement will not be processed.
Error switch: EDT.

EDT5462 SOME LINES MUST BE TRUNCATED

Meaning
A check with @CHECK (Format 2) resulted in lines or string variables whose length is longer than the maximum length allowed.
Error switch: EDT.

Response
Split or shorten lines or string variables.

EDT5463 TABULATOR POSITION NUMBER EXCEEDS MAXIMUM VALUE SUPPORTED BY HARDWARE

Meaning
The number of hardware tabulator positions defined in the @TABS statement exceeds the maximum value allowed by the terminal.
The statement will not be processed.
Error switch: EDT.

EDT5465 CANNOT CREATE ISAM KEY FROM LINE NUMBER

Meaning
When writing an ISAM file whose key position is different from standard or whose key length is too big, the ISAM key shall be built from the line number. The file will not be written.
Error switch: EDT.

Response
Write the file using KEY=DATA.

EDT5466 CANNOT IGNORE ISAM KEY

Meaning
When reading an ISAM file whose key position is different from standard, the ISAM key shall be ignored. The file will not be read.
Error switch: EDT.

Response
Read the file using KEY=DATA.

EDT5467 NO FILE OPEN

Meaning
In the @CHECK statement CODE=*FILE was specified and no file is opened.
Error switch: EDT.

Response
Specify another value for CODE operand.

EDT5468 LENGTH OF ISAM KEY CHANGED, FILE NOT WRITTEN

Meaning
If an ISAM file was opened using @OPEN statement and the coded character set was changed from UTF16 to another character set or vice versa either explicitly or implicitly by specifying a corresponding CODE operand in the @WRITE or @CLOSE statement, the file cannot be written back because this would change the length of the ISAM key.
Error switch: EDT.

EDT5469 INITIALIZATION ROUTINE MISSING

Meaning
For a user subroutine in V17 format there must exist an initialisation routine. The @RUN statement will not be processed.
Error switch: EDT.

EDT5470 VERSION ERROR IN INITIALIZATION ROUTINE

Meaning
The initialisation routine for an external statement routine or for a user subroutine does not support the version of control block (Returncode EUPVEERR is given).
The statement will not be processed.
Error switch: EDT.

EDT5471 RUNTIME ERROR IN INITIALIZATION ROUTINE

Meaning
An error occurrred in the initialisation routine for an external statement routine or for a user subroutine (Returncode EUPRTERR is given).
Error switch: EDT.

EDT5472 MINIMUM LINE NUMBER

Meaning
Using the @- statement, an attempt was made to set the actual line number to a number less than the smallest possible number (0.0001).
Error switch: EDT.

EDT5473 CCS OF STRING VARIABLES INCONSISTENT

Meaning
When a range of string variables is checked via @SEQUENCE statement (format 3), either all variables must have a Unicode character set or all must have a 7 bit or 8 bit character set.
The statement will not be processed.
Error switch: EDT.

EDT5474 MODIFIED LINE > 32768 CHARACTERS

Meaning
An edited line became too long as a result of modification with @COLUMN, @PREFIX or @SUFFIX statement. Processing of statement will be aborted.
Error switch: EDT.

EDT5475 LINE NUMBER OUT OF RANGE

Meaning
The value of a line number given indirectly or as result of an arithmetic expression is outside the valid range 0000.0001..9999.9999.
Error switch: EDT.

Response
Correct and re-enter the statement.

EDT5476 CANNOT DELETE AN ACTIVE WORK FILE

Meaning
A work file that is currently used to execute a @DO procedure (active work file), cannot be deleted.
Error switch: EDT.

EDT5477 STRING CONTENT INVALID

Meaning
The content of a string which has to be interpreted as integer in the @SET statement (format 1) or as line number in the @SET statement (format 3) is invalid.
Error switch: EDT.

EDT5478 @PARAMS STATEMENT INVALID

Meaning
A syntactical error occurred in the @PARAMS statement of a @DO procedure to be executed.
Error switch: EDT.

EDT5479 @PARAMS STATEMENT NOT FIRST LINE OF PROCEDURE

Meaning
The @PARAMS statement has to be in the first line of the @DO procedure to be executed.
Error switch: EDT.

EDT5480 LINK NAME '(&00)' NOT DEFINED

Meaning
The link name specified in a @COPY, @OPEN, @WRITE or @INPUT statement is not defined.
Error switch: EDT, DMS.

EDT5481 INVALID RECORD-FORMAT

Meaning
An attempt was made to access a file with RECORD-FORMAT=UNDEFINED or a file with RECORD-FORMAT=FIXED which has an odd record size and the coded character set UTF16. The statement will not be processed.
Error switch: EDT, DMS.

EDT5482 INVALID ACCESS TO FILE '(&00)'

Meaning
The file (&00) specified in an I/O statement could not be opened as the access was denied.
Error switch: EDT, DMS.

EDT5483 INVALID ACCESS TO MEMBER '(&00)' IN LIBRARY '(&01)'

Meaning
The element (&00) of library (&01) specified in an I/O statement could not be opened as the access was denied.
Error switch: EDT.

EDT5484 NO @FILE ENTRY DEFINED

Meaning
The file name operand is not given in an I/O statement, but neither a local nor a global @FILE entry is defined.
Error switch: EDT.

EDT5485 INPUT TOO LONG

Meaning
If a statement with indirect operands is given, the sum of the operation string length and the length of the string variable exceeds the maximum line length of 32768 characters.
Error switch: EDT.

EDT5486 WILDCARD SYMBOL MUST BE DIFFERENT TO QUOTE SYMBOLS

Meaning
An attempt was made to define a character as wildcard symbol (ASTERISK, SLASH) which is currently used as string delimiter symbol (see @QUOTE).
The @SYMBOLS statement will not be processed.
Error switch: EDT.

EDT5487 CCS '(&OO)' NOT SUPPORTED BY TERMINAL

Meaning
The communication character set given in the @CODENAME ...,TERMINAL statement cannot be used as it is not supported by the terminal.
Error switch: EDT.

EDT5488 TERMINAL NOT SUPPORTED

Meaning
The actual terminal is not supported by EDT (Unicode mode).

EDT5489 FILES MUST BE DIFFERENT

Meaning
An attempt was made to open the copy of a file for real processing (AS operand). However both files specified in @OPEN are identical.
Error switch: EDT.

EDT5490 ILLEGAL CCS NAME SPECIFIED

Meaning
An illegal CCS name was specified at the subroutine interface.
Error switch: EDT.

EDT5491 XHCS MISSING OR HAS WRONG VERSION

Meaning
The subsystem XHCS does not exist or it has a version not sufficient for EDT. EDT can not be started.
Error switch: EDT.

EDT5492 ILLEGAL RECORD FORMAT

Meaning
There are Format-B records in the library element. EDT cannot process these records.
Error switch: EDT.

EDT5493 STRING CANNOT BE CONVERTED

Meaning
Some characters of a string in the @ON statement are not defined in the character set of the line actually processed. The string cannot be converted. Processing of statement will be aborted.
Error switch: EDT.

EDT5494 'FORCE' OPERAND NOT ALLOWED

Meaning
The FORCE operand in the @CODENAME statement is not allowed for Unicode character sets.
Error switch: EDT.

EDT5495 NO FILES CORRESPONDING TO SPECIFIED OPERANDS

Meaning
No files match the file specification in @SHOW or @FSTAT statement.
Error switch: EDT.

EDT5496 INVALID VALUE FOR ATTRIBUTE (&00) IN TFT

Meaning
The file specified via a link name cannot be created, because the attribute (&00) in the Task File Table has an invalid value.
Error switch: EDT, DMS.

EDT5497 '(&00)' NOT POSSIBLE FOR PLAM ELEMENT TYPE '(&01)'

Meaning
Library elements of the type R, C, H, L, U, F or corresponding free type cannot be used with the statements @COPY, @OPEN, @WRITE or @INPUT.
Error switch: EDT.

EDT5498 CANNOT WRITE TO SYSLST

Meaning
When writing to SYSLST, an error occurred. Output will be terminated. Messages of EDT will be logged to SYSOUT from now.
Error switch: EDT.

EDT5499 WORK FILES MUST BE DIFFERENT

Meaning
Both work files specified in @COMPARE statement are identical.
Error switch: EDT.

EDT5500 STATEMENT SEQUENCE PROCESSING INTERRUPTED BY USER

Meaning
A statement sequence was specified in F-Mode dialog, in L-Mode dialog (block mode) or at the subroutine interface. The processing was interrupted by the user by means of a /INFORM-PROGRAM or /INTR command.
In F-Mode dialog, the rest of the statement sequence, which has not been processed, is output to the command line.

EDT5501 STATEMENT '(&00)' INTERRUPTED BY USER

Meaning
The processing of statement (&00) was interrupted by the user by means of a /INFORM-PROGRAM or /INTR command.

EDT5502 PROCEDURE INTERRUPTED BY USER

Meaning
The processing of a @DO or @INPUT procedure was interrupted by the user by means of a /INFORM-PROGRAM or /INTR command.

EDT5990 (&00)

Meaning
The test routine reports an error with the message (&00).
Error switch: EDT.

EDT5991 RUNTIME ERROR IN EXTERNAL STATEMENT

Meaning
The external routine reports a runtime error in the execution of the specified statement without any additional information.
Error switch: EDT.

EDT5999 (&00)

Meaning
The external routine reports a runtime error in the execution of the specified statement with the message (&00).
Error switch: EDT.

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.

EDT8002 (&00) TO EDT UNSUCCESSFULLY. RETURNCODE = X'(&01)'

Meaning
An error has occured during the dynamic loading of EDT. The macro (&00) rejected the loading with returncode (&01).
If the retuncode is X'0C010104', then a call of EDT under service user id is only possible if EDT subsystem is loaded.

EDT8003 NO VIRTUAL MEMORY AVAILABLE

Meaning
The initialisation of EDT could not be processed, as there is not enough memory for the EDT internal data.

Response
Release virtual memory.

EDT8005 ERROR ON EDT INITIALIZATION

Meaning
An error occurred during initialisation of EDT.

EDT8006 ERROR ON INSTALLATION OF EDT

Meaning
The EDT installation items are inconsistent, or EDT cannot be started under current operating system version.

Response
Check and correct the installation of EDT.

EDT8100 EDT INTERRUPTED BY USER

Meaning
This message serves as information for SDF-P procedures.
EDT is loaded, but has been interrupted by @SYSTEM (without operand) or by an explicit K2.

EDT8101 USER TERMINATED EDT ABNORMALLY

Meaning
The user terminated EDT by the statement @HALT ABNORMAL.

EDT8200 STXIT ROUTINE FOR RUNOUT ACTIVATED

Meaning
The end of the program run time has been reached, therefore EDT is terminated.

EDT8292 UNRECOVERABLE READ ERROR. PROGRAM ABORTED

Meaning
An unrecoverable error occurred while reading from SYSDTA or from terminal.
EDT is terminated.

EDT8293 UNRECOVERABLE WRITE ERROR. PROGRAM ABORTED

Meaning
An unrecoverable error occurred while writing to SYSOUT. EDT is terminated.

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)

Meaning
A program interruption of event class "program error" or "unrecoverable program error" occurred at location (&00). Detailed information about the error cause is given by the interrupt weight (&01).

Response
Contact the system administrator.

14 Logistics

This section describes the requirements and procedures for the installation and start-up of EDT V17.0A.

14.1 Software requirements

EDT V17.0A has been released for operating system versions as of OSD-BC V6.0. To provide support for Unicode mode, the following subsystems must be installed and running in the system:

XHCS-SYS	V2.0 A14 or higher
CRTE-BASYS	V1.6 A10 or higher
VTSU	V13.2 A05 or higher
TIAM	V13.1 C03 or higher
SYSFILE	V15.0 C00 or higher

The following are also required for individual functions:

JV	V14.0 or higher if job variables are to be used
SDF	V4.6 or higher if the @SDFTEST statement or the T statement code are to be used
SDF-P	V2.3 or higher if S list variables are to be used
POSIX	V6.0 A39 or higher if POSIX files are to be processed

The user-friendly handling of Unicode files in interactive mode is only possible when working with a terminal emulation which is able to process Unicode characters, e.g. under Windows the emulation MT9750^{TM} V7.0 or higher.

14.2 Scope of delivery

The product EDT V17.0A comprises the following release items:

Release item	Content
SYSPRG.EDT.170.EDTU	Phase for EDT (start in Unicode mode)
SYSPRG.EDT.170	Phase for EDT (start in compatibility mode)
SYSLNK.EDT.170	EDT module library
SYSLNK.EDT.170.INIT	Library containing the EDT initialization module
SYSLIB.EDT.170	User macro library
SYSMES.EDT.170	System message file
SYSSSC.EDT.170	Subsystem declaration for SSCM
SYSSII.EDT.170	Structure and installation information
SYSRMS.EDT.170	Correction storage for RMS
SYSREP.EDT.170	REP file
SYSFGM.EDT.170.D	Release Note (German)
SYSFGM.EDT.170.E	Release Note (English)
SYSSDF.EDT.170	Syntax file for SDF (START-EDT / START-EDTU)
SINPRC.EDT.170	Procedure for installing a private version
SYSSMB.EDT.170	Symbol information for diagnosis with DAMP

Note

The SYSNRF files supplied in the past are no longer required. Instead the REPs for EDT are indicated by the corresponding REP identifier.
(S for *Selectable Unit* or U for *Selectable Unit Optional*).

The SYSACF file for assigning the alias name $.EDTLIB is no longer supplied. Instead of incorporating the SYSACF file in the system alias catalog, the same result can be achieved using the system administrator command:

```
/ADD-ALIAS-CATALOG-ENTRY ALIAS-FILE-NAME=$.EDTLIB,FILENAME=$.SYLNK.EDT.170
```

If installation is performed using IMON then the phase for starting in compatibility mode (SYSPRG.EDT.170) is stored under the name EDT in the installation ID.

14.3 Product structure

The file SYSSII.EDT.170 contains the structure and installation information. Logical names (*Logical ID*) are assigned to the product components. IMON can use these to identify the current installation location.

Release item	Logical ID
SYSPRG.EDT.170.EDTU	SYSPRG.EDTU
SYSPRG.EDT.170	SYSPRG
SYSLNK.EDT.170	SYSLNK
SYSLNK.EDT.170.INIT	SYSLNK.INIT
SYSLIB.EDT.170	SYSLIB
SYSMES.EDT.170	SYSMES
SYSSSC.EDT.170	SYSSSC
SYSSII.EDT.170	SYSSII
SYSRMS.EDT.170	SYSRMS
SYSREP.EDT.170	SYSREP
SYSFGM.EDT.170.D	SYSFGM.D
SYSFGM.EDT.170.E	SYSFGM.E
SYSSDF.EDT.170	SYSSDF
SINPRC.EDT.170	SINPRC
SYSSMB.EDT.170	SYSSMB
*DUMMY.EDTSTART	SYSDAT.EDTSTART

The module library SYSLNK.EDT.170 contains the following modules:

Module	Function	Unicode /compatibility mode
EDTSTRT	Drivers for START-EDT and START-EDTU	U/C
EDTU	Main module for EDT	U
EDT	Main module for EDT	C
EDTXCODIR	Dynamically loadable module for CODE statement	C
EDTXPKEY	Dynamically loadable module for PKEY statement	C
EDTSSLNK	DSSM dynamically loadable module	U/C
IEDTGLE	Link and load module for EDT as a subroutine	U/C
CODTAB	Code table for CODE statement	C
EDTCON	Connection module for EDT, switch operating mode	U/C
EDCNATA	Dynamically loadable module for national applications	C

The module library SYSLNK.EDT.170.INIT contains the following module:

Module	Function	Unicode /compatibility mode
IEDTCRT	Module for initializing EDT and switching between Unicode and compatibility mode.	U/C

Details concerning the components which are only required for compatibility mode are not described any further here. The corresponding descriptions can be found in the EDT V16.6B User Guide [2].

14.4 Installation

EDT can be installed publicly using the SOLIS procedure or can be installed by a user as a private installation under any required user ID.

14.4.1 Public installation

By default, the product is installed using the SOLIS procedure. In turn, this requires the IMON installation monitor. EDT V17.0 can no longer be used in systems in which IMON is not present.

In existing programs which call EDT via the L mode subroutine interface, the library name $EDTLIB is present as a fixed program element. This type of program can only work with EDT V17.0 in compatibility mode. To ensure that these programs can continue to run unchanged, it is either necessary to save a copy of the library SYSLNK.EDT.170 under the name $TSOS.EDTLIB or make a corresponding entry in the global system alias catalog (see manual Guide to Systems Support [13] section ACS: Alias catalog system).

The REP file SYSREP.EDT.170 must be shareable (USER-ACCESS=SPECIAL). Only then are the corrections loaded on the dynamic loading of EDT.

For performance reasons, it is also possible to load EDT as a subsystem (see below).

Start procedure: EDTSTART

For each publicly installed EDT version, IMON can be used to install an EDT start procedure which is valid for all user IDs. The installation location of the procedure file can be chosen freely as required (see also the section on the EDT start procedure). The logical identification SYSDAT.EDTSTART is defined for this file in the SYSSII file.

The /SET-INSTALLATION-PATH command can be used to inform the installation monitor of the installation file name. EDT retrieves this information using the IMON function GETINSP and the defined path is set instead of $.EDTSTART. If no file is assigned to the logical ID SYSDAT.EDTSTART then EDT uses the EDT start procedure $.EDTSTART if this exists.

EDT as subsystem

EDT consists of three subsystems: EDTCON, EDTU and EDT. All the EDT subsystems run independently of the addressing mode, i.e. they can run with 24-bit or 31-bit addressing. The decision whether to load a subsystem into the upper or lower address space therefore depends on the addressing mode applicable to the most frequent users of the EDT subroutine interface.

The EDTCON subsystem (which consists of the modules EDTCON, IEDTGLE and EDTSSLNK) is loaded into the lower address space. In EDT V17.0A, EDTCON is simply an adapter which supplies the previous EDT entries in compatible form and dynamically loads its own initialization module IEDTCRT from the library SYSLNK.EDT.170.INIT into the user's address space. However, the EDTCON module is no longer supplied as an OM but as an LLM.

The EDTU and/or EDT subsystems can be loaded into the upper address space. The EDTU subsystem contains the modules which are required for EDT to run in Unicode mode, while the EDT subsystem contains the equivalent modules for compatibility mode. Depending on the intended type of utilization, it may make sense to preload only one of the two subsystems. When the operating mode is switched, the other subsystem would then be dynamically loaded as a private copy. However, it is only possible to preload these two subsystems if the EDTCON subsystem has already been preloaded.

If EDT is to run as a main program or subroutine in 24-bit addressing mode then either the IEDTCRT module establishes a connection to an EDT system loaded in the lower address space or EDT is dynamically loaded privately in the lower address space.

The mechanism for switching between the EDT operating modes is also implemented in IEDTCRT. This means that depending on whether EDT is called as a main program via /START-EDTU or /START-EDT or, equally, on the employed version of the subroutine

interface, processing branches either to the EDTU subsystem (Unicode mode) or the EDT subsystem (compatibility mode). IEDTCRT functions are also used when the operating mode is changed explicitly or implicitly due to user input.

The EDTCON subsystem can be preloaded with

```
/START-SUBSYSTEM SUBSYSTEM-NAME=EDTCON,SYNC=*YES
```

It is then possible to start the subsystems EDTU and/or EDT using

```
/START-SUBSYSTEM SUBSYSTEM-NAME=EDTU
```

or

```
/START-SUBSYSTEM SUBSYSTEM-NAME=EDT
```

in any desired order.

14.4.2 Private installation

The file SINPRC.EDT.170 contains a procedure which can be used to install a private version under any user ID. Private versions should only be used for test purposes and should not lead to the coexistence of two EDT versions.

Before this procedure is called, at least the following files must be available under the installation ID for the private EDT version:

- SYSPRG.EDT.170 and SYSPRG.EDT.170.EDTU (starter phases)
- SYSLNK.EDT.170 (module library)
- SYSLNK.EDT.170.INIT (module library containing the EDT initialization routine)
- SYSREP.EDT.170 (REP file)

For reasons of compatibility, a file named EDT is also used as the starter phase for compatibility mode (as in a public installation) if no file named SYSPRG.EDT.170 is present under the installation ID. However, it is advisable to use the name SYSPRG.EDT.170 since this is the only way of creating multiple private installations of different EDT versions under a single installation ID.

The procedure can be called under the installation ID or TSOS:

```
/CALL-PROCEDURE FROM-FILE=SINPRC.EDT.170, -
/PROCEDURE-PARAMETERS=(USERID=userid,ORIG=Y|N, -
/EDT=edt,EDTU=edtu,EDTLIB=edtlib,INILIB=inilib,
/REPFILE=repfile)
```

The procedure parameters can be specified directly or via *parameter prompting*. The parameters EDT, EDTU, EDTLIB, INILIB and REPFILE only need to be specified if ORIG=N has been specified.

userid Installation ID under which the private EDT version is to be installed.

ORIG=Y Specifies that the provided original files (under the installation ID) are to be modified.

ORIG=N Specifies that copies of the provided original files are to be created and that the modifications are to be made in the copies only.

edt Name of the copy for the private starter phase for compatibility mode (if the original files are not to be edited).

edtu Name of the copy for the private starter phase for Unicode mode (if the original files are not to be edited).

edtlib Name of the copy for the private module library (if the original files are not to be edited).

inilib Name of the copy for the private module library containing the EDT initialization module (if the original files are not to be edited).

repfile Name of the copy for the private REP file for EDT (if the original files are not to be edited).

The private EDT version can be started with the command

```
/START-PROGRAM FROM-FILE=$userid.name-private-starterphase
```

The installation procedure freezes all the references to the logical names defined in IMON for the private version. This is particularly important if a SYSDAT.EDTSTART assignment has been defined. Any EDT start procedure defined centrally in this way is therefore not executed by the private version.

Programs that want to call this private EDT version must either link the connection module IEDTGLE from the created module library or dynamically load it from this library using the ENTRY name IEDTGLE. When doing this it is essential to specify the parameter SHARE=NO in the BIND macro.

Programs that want to call the private EDT version as a subroutine and dynamically load EDT from $.EDTLIB using a BIND (not LINK) macro call can assign a private module library with the following command:

```
/SET-FILE-LINK LINK-NAME=BLSLIBnn,FILE-NAME=library
```

where nn=00..99. This procedure is only possible if EDT is not loaded as a subsystem.

Glossary

@DO procedure

A @DO procedure is an EDT procedure which is stored in a work file. It can be run by means of a @DO statement. @DO procedures provide a number of statements used for runtime control. At call time, it is possible to pass parameters to these statements which can also be nested.

@INPUT procedure

An @INPUT procedure is an EDT procedure which is stored in a file or library element. It can be run by means of an @INPUT statement. The runtime control statements are not (directly) available in @INPUT procedures. These procedures cannot be nested and it is not possible to pass any parameters. It is, however, possible to call @DO statements.

Batch mode

Batch mode is the EDT operating mode in which no terminal is present. EDT can then only operate in L mode.

Character set

EDT V17.0A makes it possible to process texts in all the character sets made available by XHCS. In addition to the character sets supported in EDT V16.6B and compatibility mode, the character sets supported in Unicode mode include the three Unicode character sets, ISO character sets and additional 7-bit character sets.
In each work file it is possible to configure a different character set so that texts in different character sets can be processed in parallel. A communications character set is configured to permit communication with a terminal.

Communications character set

Character set used by EDT in Unicode mode to communicate with the terminal. This can be different from the character set used by the current or any other work file. The communications character set is usually optimally suited to the capabilities offered by the terminal.

Compatibility mode

Compatibility mode is an EDT V17.0A operating mode. Compatibility mode provides the full functionality of EDT V16.6B. However, the extended functions of EDT V17.0A cannot be used. For example, it is not possible to process Unicode files and the record length continues to be restricted to 256 bytes. Under some circumstances, there may be an implicit switchover to Unicode mode if a Unicode file is read or an explicit switchover may occur if a @MODE statement is entered.

Data window

Field in the work window in which the current work file is displayed. The work file records are output in the data window's screen lines.

Delimiter characters

Literals are usually enclosed by the delimiter character `apostrophe` (or single quote, default value '). When a search is performed using the @ON statement, it is possible to use a special delimiter character, namely the `quotation mark` (default value “). This specifies that a string is only recognized as a hit if it is delimited by text delimiters. The text delimiter characters consist of a configurable set of characters which, by default, include the space character, parentheses etc.

EDT procedures

An EDT procedure is a sequence of entries, statements and/or records sent to EDT and stored in a file (@INPUT procedure) or work file (@DO procedure).

EDT start procedure

The EDT start procedure is a special @INPUT procedure which (if present) is run when EDT is started.

Interactive mode

Interactive mode is the EDT operating mode in which a terminal is present. This is the only mode in which EDT can operate in F mode.

EDT statement symbol

The EDT statement symbol is a special character used to identify statements. By default, this is the character @ which is therefore consistently used in this document.

EDT variables

EDT variables are containers which can be used to store values across work file boundaries. EDT variables are only valid for the current EDT session. There are three types of variables which can be assigned the corresponding values. 21 variables of each variable type are available.

- Integer variables (#I0..#I20)
- String variables (#S0..#S20)
- Line number variables (#L0..#L20)

Full screen mode (F mode)

Full screen mode (F mode) is an EDT work mode. In F mode, the entire screen is available as a work window for the entry of data and statements. It is possible to switch from F mode to L mode. EDT can only operate in F mode if it is in interactive mode.

Line mode (L mode)

Line mode (L mode) is an EDT work mode. In L mode, files are processed line-by-line, that is to say that in dialog operation EDT only outputs one line (the current line) at a time or only reads one line (in both batch and dialog operation) from SYSDTA. This line may contain records or statements and is processed as soon as it has been read in.

Line number

A line number is assigned to every record in a work file. This line number uniquely identifies the record. A line number is the current line number in which data is entered in L mode.

Operating mode

Since it was not possible to implement the extensions required for Unicode support in a way which would ensure compatibility, a new EDT operating mode has been introduced. EDT V17.0A can therefore be used in two modes: Unicode mode and compatibility mode.
In Unicode mode, a range of extensions are available. Most importantly, it is possible to process Unicode files (only) in Unicode mode.
However, this mode is not compatible with EDT V16.6B in all respects.
In contrast, compatibility mode provides the full functionality of EDT V16.6B. However, the extensions are not available in this mode.
By default, EDT V17.0A is started in compatibility mode. A new statement is available to start it in Unicode mode.

Operating types

A distinction is made between two type of operation depending on whether or not a terminal is present. In interactive mode, a terminal is present whereas it is not in batch mode.

Record mark

Every record in a work file can be flagged with a record mark which is invisible to users. These marks can be set, queried and deleted using statements and statement codes. Once marked, records can, for example, be copied or deleted.

Screen dialog

The statement @DIALOG – which can only be entered at the subroutine interface or by SYSDTA – is used to switch EDT to screen dialog. In screen dialog, the preceding read operation is interrupted and EDT reads its input from the terminal in F mode (or in L mode after the entry of @EDIT). The screen dialog can be exited again with @HALT, @END, @RETURN or K1. EDT then continues the interrupted read operation.

Screen line

Lines in the data window in which the records of the current work file are output.

Statement

Inputs to EDT take the form either of records or statements. Statements are used either to activate EDT functions or make parameter settings. In order to distinguish between statements and records, statements must be entered in line mode with the EDT statement symbol. In F mode, statements are entered in the statement line There are also statement codes which are entered in the statement code column.

Statement buffer

EDT saves the most recent statements entered in F mode in a buffer. These statements can then be retrieved from this buffer.

Statement code

Statement codes are 1 character long statements which can be entered in the statement code column in F mode.

Statement code column

The statement code column is a field in the work window in which statement codes can be entered.

Statement line

A work window field in F mode. Entries in the statement line are interpreted as statements. The EDT statement symbol can normally be omitted.

Substitute character

When strings are converted from one character set to another, it is possible that characters in the source character set may not be present in the target character set. In such cases, the substitute character is used (if it has been defined). If no substitute character has been defined then conversion is usually rejected.

Unicode mode

Unicode mode is an EDT operating mode. The extended EDT V17.0A functions are only available in Unicode mode, i.e. only in this Unicode mode is it possible to process Unicode files, can records exceed 256 bytes in length, are local character sets available etc.
However, this mode is not compatible with EDT V16.6B in all respects. In particular, there are incompatibilities in terms of the subroutine interface. However, a large number of inconsistencies have been eliminated.

Unicode substitute representation

In both statements in literals and in data, EDT permits the substitute representation of Unicode characters through the specification of the associated `UTF16` code. This makes it possible to enter all the (supported) characters via an escape character even if the character set for the statement or the data is not a Unicode character set.

User statement symbol

A user statement symbol is a special character which identifies user statements which are executed using external statement routines.

Wildcards

Wildcards are placeholders for groups of characters in a search string. Here, the `asterisk` (default value `*`) stands for a string of any length (including an empty string) and the `slash` (default value `/`) for precisely one character.

Work file

In EDT, data is always entered and processed in a work file. In work files, it is possible, for example, to insert, edit and delete data. The content of work files can be displayed on screen. If it is necessary to process the content of a file (DMS file, library element or POSIX file) then this content must first be transferred to a work file. After processing, the content of a work file can be written back to a file.
EDT is able to manage 23 work files. The work files are organized into records to which line numbers are assigned.

Work file, active

An active work file is a work file which contains a @DO procedure which is currently being executed. If there are nested @DO procedures then multiple work files may be active.

Work file, current

A single work file is the current work file. Data is entered in this work file and statements are effective within it. In F mode, a section of the current work file is displayed on the screen.

Work file, empty

An **empty** work file is a work file which contains no records. However, an empty work file may also contain properties which do not correspond to the initial state, for example it can be considered to be in use or occupied or linked to a file. A work file is only reset to its original state following a @DELETE statement (format 2) or other statements which completely delete work files either implicitly or explicitly.

Work mode

EDT provides two work modes for data processing: line mode (L mode) and full screen mode (F mode).
In L mode, only one screen line is available for the entry of data and statements at any time.
In F mode, the entire screen is available for the entry of data and statements.
The work modes should not be confused with the EDT operating modes (compatibility mode and Unicode mode). While the work modes relate to differences in the way data is displayed and processed, the operating modes represent different EDT environments with a restricted or extended function scope.

Work window

In F mode, the current work file is displayed on the screen. In this case, the screen is subdivided into fields with different functions. Alongside the data window in which the content of the current work file is displayed, the work window contains a statement line and a statement code column together with other elements.

Related publications

The manuals are available as online manuals, see *http://manuals.fujitsu-siemens.com*, or in printed form which must be paid and ordered separately at *http://FSC-manualshop.com*.

[1] **EDT V17.0A UNICODE Mode** (BS2000/OSD)
Subroutine Interface
User Guide

[2] **EDT V16.6B** (BS2000/OSD)
Statements
User Guide

[3] **EDT V16.6** (BS2000/OSD)
Subroutine Interface
User Guide

[4] **EDT-ARA** (BS2000/OSD)
Additional Information for Arabic
User Guide

[5] **EDT-FAR** (BS2000/OSD)
Additional Information for Farsi
User Guide

[6] **SDF** (BS2000/OSD)
Introductory Guide to the SDF Dialog Interface
User Guide

[7] **SDF-P** (BS2000/OSD)
Programming in the Command Language
User Guide

[8] **XHCS** (BS2000/OSD)
8-Bit Code and Unicode Processing in BS2000/OSD
User Guide

[9] **JV** (BS2000/OSD)
Job Variables
User Guide

[10] **BS2000/OSD-BC**
Commandes Volume 1-5
User Guide

[11] **BS2000/OSD-BC**
Commands Volume 6, Output in S Variables and SDF-P-BASYS
User Guide

[12] **BS2000/OSD-BC**
Executive Macros
Benutzerhandbuch

[13] **BS2000/OSD-BC**
Introductory Guide to System Support
User Guide

[14] **LMS** (BS2000/OSD)
SDF Format
User Guide

[15] **POSIX** (BS2000/OSD)
POSIX Basics for Users and System Administrators
User Guide

[16] **POSIX** (BS2000/OSD)
Commands
User Guide

[17] **ASSEMBH** (BS2000/OSD)
Reference Manual

[18] **ASSEMBH** (BS2000/OSD)
User Guide

Index

X

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *...@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at *http://ts.fujitsu.com/*...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *...@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.