1 Preface

EDT (which is simply an abbreviated form of “editor”) is used to edit files. It can edit SAM and ISAM files, elements of program libraries and POSIX files.

With EDT, the user can

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

EDT offers the following facilities for editing data:

1. Virtual processing of files and library elements in interactive mode

   a) creation and editing in the user address space

   b) writing and storing a file or a library element from the user address space to disk or tape. The main advantages of processing in the user address space are that

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

2. Real processing of files in interactive mode
   The files can be processed directly on the disk.

3. Processing of files and library elements using EDT procedures
   File processing operations which have to be executed frequently in the same or a similar manner can be programmed as EDT procedures.

4. Processing in batch mode
   Although EDT was designed as an interactive program, it can also be used for the virtual or real processing of files and library elements in batch mode.

EDT can

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

1.1 Structure of the EDT documentation

The complete documentation for EDT comprises the manuals:

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

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

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

The "Ready Reference" contains summary descriptions of all EDT statements.

1.2 Target groups for the EDT manuals

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

This manual on the EDT subroutine interfaces addresses experienced EDT users and programmers wishing to use the many features of EDT in their own programs.

For calling EDT as a subroutine, familiarity with the major BS2000 commands and with EDT and its statements as well as knowledge of Assembler, COBOL and/or C are absolutely essential.

1.3 Structure of the "EDT Subroutine Interfaces" manual

This manual deals solely with the subroutine interfaces of EDT.

It includes the following chapters:

- **Introduction**

  Overview of the functions and application options and brief description of the control blocks.

- **EDT as a subroutine**

  Description of the functions (with calls and return codes), format of the five control blocks, the include elements for C programming, and examples for calling EDT from a COBOL, Assembler or C program.

- **External statement routines**

  Description of the transfer of statements written by the user; special application as a statement filter.

- **Calling a user program**

  Description of the loading and unloading of a user program; overview of the information passed to the user program; description of the routines for editing lines.

- **L mode subroutine interface**

  Description of the compatible subroutine interface of the L mode. This interface is supported for reasons of compatibility only and should not be used in new applications.

A detailed description of EDT, including the wide range of EDT statements and messages, can be found in the companion manual:

EDT (BS2000) V16.6
Statements
User Guide

Summary descriptions of all EDT statements are contained in:

EDT (BS2000) V16.6
Statement Formats
Ready Reference

References to other publications are given in the text in the form of abbreviated titles. The complete title of each publication referred to via a number can be found under "Related publications" after the appropriate number. This is followed by a brief note on how to order manuals.

README file

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

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

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

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

1.4 Changes since the last version of the manual (EDT V16.5)

Extension of the subroutine interface

The calling program can instigate deletion of the EDT copy buffer.

IEDTDEL interface

In addition, the calling program can determine whether the work file is to be marked as modified when a record is written.

IEDTPUT interface

C include files

The following include files are available for programming the subroutine interface using C:

iedglcb.h

iedupcb.h

iedambc.h

iedparg.h

iedparl.h

1.5 Notational conventions

The following notational conventions are used in this manual:

“quotes“	Chapter/section titles and words to be emphasized.
␣	Blank.
<u>Underscore</u>	Default.
[Number]	Reference to a manual in the “Related publications” section.
$\boxed{\text{Key}}$	Symbolizes a key on the keyboard.
$\boxed{\mathbf{i}}$	Indicates additional information.

2 Introduction to the EDT subroutine interface

The subroutine interface of EDT offers the following functions and application facilities:

- Calling EDT with return of an information block (INFO function).
- Calling EDT for execution of an EDT statement sequence (CMD function).
- Calling EDT for processing of a file with logical record access functions:
  - read a record
  - read a marked record
  - write a record
  - mark a record
  - delete a record
  - modify a record index
  - read the work file status

Brief description of the control blocks

If EDT is called as a subroutine, data is transferred with the aid of five control blocks, which must be defined in the user program. Macros are available for the definition of these blocks using the Assembler programming language, and include files are available for definition if the programming language C is used.

EDTGLCB: Global Control Block

The global control block contains the data fields which are required at each interface between a user program and EDT.

EDTUPCB: subroutine control block

This control block is used with the CMD function to pass the necessary parameters for the default values to EDT.

EDTAMCB: access method control block

The EDTAMCB control block is used to pass to EDT all data which is necessary for the logical record access functions.

EDTPARG / EDTPARL: parameter settings (global/local)

For the function "Read file status", EDT returns the information about the file status in these control blocks.

For a more detailed description of the control blocks see section “EDTGLCB - Global control block” on page 53ff.

3 Calling EDT as a subroutine

3.1 Linking the user program to EDT

The IEDTGLE module from the module library SYSLNK.EDT.166 (alias name as of BS2000/OSD V1.0: $EDTLIB) is linked in the user program (main program) using a V-type constant.

The module IEDTGLE

– contains all entry point addresses for the use of the various functions

– calls the reentrant part of EDT, using the BIND macro

– saves the entry point address of EDT in the global control block EDTGLCB.

The BIND macro is executed for the first call only. Subsequent calls take the entry point address from the EDTGLCB module.

The IEDTGLE module is reentrant. It executes the branch to EDT and transfers the parameter list without modification.

3.1.1 Calling EDT

EDT is called in accordance with the standard rules for program linkage. It can also be called by higher-level programming languages. Before EDT is called, the registers must be loaded as follows:

Register	Data area
(R1)	A(PARAMETERLIST)
(R13)	A(SAVEAREA)
(R14)	A(RETURN)
(R15)	V(ENTRY)

PARAMETERLIST

This data area must be created by the user.

The parameter list must contain the addresses of all control blocks and defined data fields from which EDT can extract the necessary data (such as statement sequences, message texts, etc.).

The parameter list depends on the function of the call (INFO, CMD, EXE function). The parameters which must be provided are shown in the tables in the sections describing the various functions.

SAVEAREA

A register save area (18 words, DC 18F’0’), which must be created by the caller and in which EDT saves the registers.

RETURN

The return address in the calling program. The program will be continued at this address when EDT is terminated.

ENTRY

For each function, the IEDTGLE module contains a separate entry point address (ENTRY):

Entry point	**Function**	**Page**
IEDTINF	Read the EDT version numbe	16
IEDTCMD	Execute an EDT statement or statement sequence	18
IEDTEXE	Execute an EDT statement or statement sequence	24
IEDTGET	Read a record	31
IEDTGTM	Read a marked record	35
IEDTPUT	Write a record	40
IEDTPTM	Mark a record	42
IEDTDEL	Delete the copy buffer or a record range	45
IEDTREN	Modify the record index	47
IEDTGET	Read the work file status	49

Initializing EDT

For all functions except IEDTINF, EDT must always be initialized by means of the IEDTCMD function before the call is issued. It is not necessary to pass a statement sequence.

Call parameters

Before calling a function, the user must specify values for the control block fields of EDTGLCB (call parameters). Which fields have to be supplied explicitly depends on the function to be used. These fields are listed in the table "Call parameters" in the sections describing the various functions.

Return parameters

When the function has been completed, EDT places return parameters in defined fields of EDTGLCB, e.g.

– return codes

– data of a message text

The return parameters are listed in the table "Return parameters" in the sections describing the various functions.

Return codes

In the global control block EDTGLCB, EDT passes the following return codes:

– **main value** in field EGLMRET
  This limits the error class.

– **1st subcode** in field EGLSR1
  This contains information for a precise definition of the error.

– **2nd subcode** in field EGLSR2
  This field always contains X’00.

The possible return codes and their meanings are contained in control block EDTGLCB (see section “EDTGLCB - Global control block” on page 53ff).

Return codes for the INFO function, CMD function and EXE function

The prefix EUP identifies the return codes which are passed if EDT is called as a subroutine for reading the version number or for executing a statement sequence.

Return codes for the record access functions

The prefix EAM identifies the return codes which are passed if a record access function of EDT is called.

3.1.2 Memory reorganization in subroutine applications

Applications using the EDT subroutine interfaces may store the record addresses in the main program or in a user routine for subsequent use, which is why reorganization is not desired.

To ensure upwardly compatible support for such applications, reorganization is only performed on explicit request. The EGLREOR flag in the EGLINDB indicator byte of the EDTLCB control block serves this purpose.

This flag is evaluated on the calls IEDTCMD, IEDTEXE, IEDTPUT and IEDTDEL. Its setting applies to the current call only. Reorganization can thus be suppressed during the runtime of a user program (@RUN).

3.1.3 Interrupt handling

In EDT, STXIT routines are defined for the following events: ESCPBRK, program check, unrecoverable program error, message to the program, program runtime exceeded, and EXIT-JOB MODE=ABNORMAL.

At the IEDTCMD interface you can specify whether STXIT routines are to be activated. For this purpose, the EGLSTXIT flag in the EGLINDB byte of the EDTGLCB control block must be set explicitly.

The STXIT routine for ESCPBRK is not provided in batch mode if task switch 5 (procedure mode) is set and at the subroutine interface of the L mode.

In F mode and in L mode, the EDT run can be interrupted via @SYSTEM or $\boxed{\text{K2}}$. If the STXIT routine for ESCPBRK is active, R1 contains the address at which the EDT run was interrupted. The contents of registers R0, R2 - R4 and R15 are not relevant to the interrupted task.

You can return to the interrupted work mode of EDT by means of the RESUME-PROGRAM command. In F mode, the screen with the original contents is redisplayed completely. EDT output to SYSOUT ($\boxed{\text{K2}}$ following %PLEASE ACKNOWLEDGE) is aborted.

Processing of the current statement is continued up to the logical end of a work step, then further processing is aborted. For example, upon $\boxed{\text{K2}}$ during @COPY 1-10(0) TO 1 5-20(1) TO 40 abortion takes place after @COPY 1-10(0) TO 1 has been executed.

Upon interruption of the EDT run in F mode the rest of the statement chain is not executed.

Upon interruption of the EDT run in L mode the STXIT routine of EDT closes, among other things, the files opened for @READ, @GET and @INPUT. If no @INPUT or procedure file is active, the current statement symbol is output on the screen. If EDT has not yet fully processed the lines of an @INPUT file or input block (BLOCK mode) at the time of the interrupt, the lines not yet processed are lost.

If START-PROGRAM or LOAD-PROGRAM is entered or procedures containing these commands are started during the interrupt, EDT is unloaded.

If the event "program runtime exceeded" occurs (EDT runtime greater than the CPU-LIMIT value specified in the START-PROGRAM command), a message to this effect is output to SYSOUT and, in batch mode, EDT is abnormally terminated.

If the interrupt event PROCHK (program check) or ERROR (unrecoverable program error) occurs and the EDT data area is still addressable, a message is output in which the program counter and the interrupt weight are indicated. In interactive mode, either EDT is abnormally terminated or an attempt is made (e.g. on a data error in L mode) to remove the invalid data by deleting the current work file. If this is unsuccessful, TERM is issued and a memory dump requested.

3.1.4 Support for extended character sets (XHCS)

Via the EDT subroutine interfaces, data can be imported to EDT. This data may be coded in a specific extended character set (CCS). In this case, EDT must be informed of the character set name (CCSN) of the data supplied. This information can be provided via IEDTCMD and IEDTEXE through the @CODENAME statement.

Conversely, the information on the currently applicable CCSN in EDT may be relevant to the main program and also to user routines. To this end, the CCSN in the EPGCCSN field of the IEDTPARG control block can be evaluated.

Details on extended character sets (CCS) and XHCS can be found in the manuals "EDT Statements" [1] and "XHCS" [11].

3.1.5 EDT in the XS environment

EDT is executable in 24-bit and 31-bit addressing modes.

The following point should be borne in mind:

> When a program is called by EDT, it is the program’s responsibility to set the correct processing mode. On leaving the program, the addressing mode valid prior to the call must be restored.

If a user program that can only run in 24-bit addressing mode is to be executed, EDT must be called as follows:

```
START-PROGRAM *MOD($EDTLIB,EDTC)
```

or, as of BS2000/OSD V2.0, by means of

```
START-EDT PROGRAM-MODE=24 (or EDT PROG-MODE=24)
```

EDT then runs in 24-bit addressing mode.

3.2 Statement functions

The statement functions can be used to:

- obtain information on the version number of EDT and on the number of memory pages for the static file area
- transfer a statement or sequence of statements to EDT.

The following functions are available:

Function	**Entry point address**
Read the version number of EDT	IEDTINF
Execute EDT statements	IEDTCMD
Execute EDT statements without a screen dialog	IEDTEXE

For a description of the individual functions see section “IEDTINF - Read EDT version number” on page 16ff.

3.2.1 IEDTINF - Read EDT version number

At the IEDTINF interface

– EDT is loaded when first called without a version number

– the user is informed of the version number of the loaded EDT program

– it is possible to specify a version number if EDT is installed under IMON (as of BS2000/OSD V2.0). EDT is then loaded only if the specified version exists

EDT is loaded without initialization when first called.

1. Call without version selection
   The following information is passed in the EDTGLCB control block:
   – full version number of EDT in the EGLRMSGF field (printable)
   – number of memory pages for the static file area (words) in the EGLINFM field
   – return code EUPRETOK in the EGLMRET field if the call is successful; if not EUPEDERR or EUPPAERR

2. Call with version selection
   The version of EDT to be loaded if specified in the EGLRMSGF field. The length of the version entry must be specified in the EGLRMSGL field.

Length of version number:	9-12	
Format of version number:	EDT Vaa.a[d[ii]]	where a and i are numbers and d is a letter

A version specification is recognized on the basis of a length entry in the EGLRMSGL field; the value must be in the permitted range between 9 and 12. If no version is specified and a number of versions are available, the EDT version set in SET-PRODUCT-VERSION or the latest version of EDT is loaded.
If the specified version cannot be loaded, the return code EUPVEERR is set in the EGLMRET field and no version is loaded. If the STD version can be found, it is entered in the EGLRMSGF field. If the STD version cannot be found, the subreturn code EUPVE04 is set in the EGLSR1 field.
If the call is successful, the return code EUPRETOK is set in the EGLMRET field (EDTGLCB).

Call

The following information is required:

– the entry point address IEDTINF must be specified
– the parameter list with the address of EDTGLCB must be created
– the call parameters must be placed in the fields of control block EDTGLCB

Overview table

Control blocks: see section “EDTGLCB - Global control block” on page 53ff.

Entry point address	:	IEDTINF

Parameter list	:	A(EDTGLCB)

Call parameters		**Return parameters**	
EDTGLCB:	EGLUNIT EGLVERS EGLINDB (EGLMSG)	EDTGLCB:	EGLRETC EGLRMSG EGLINFM

Return codes

EGLMRET	**EGLSR1**
EUPRETOK EUPEDERR EUPPAERR EUPVEERR	EUPOK00 00 EUPPA04 EUPVE00 EUPVE04

The fields EGLMRET and EGLRS1 are located in control block EDTGLCB. The meanings of the return codes are described in more detail in section “EDTGLCB - Global control block” on page 53ff.

Call in the C program

Include files required:

– #include <stdio.h>
– #include "iedglcb.h"

The EDTGLCB control block is declared and initialized as follows:

```
extern void iedtinf();
```

```
struct iedglcb_mdl iedtglcb=IEDGLCB_INIT;
```

In the C program, too, the IEDTINF function is called at the address of EDTGLCB:

```
iedtinf(&iedtglcb)
```

3.2.2 IEDTCMD - Execute EDT statements

With this call, a statement or a statement sequence with a length of up to 256 characters (plus a 4-byte record length field) is passed to EDT.

With the exception of @EDIT, all EDT statements which may be used in both F mode and L mode are permitted. The @EDIT statement is only permitted in the form @EDIT ONLY.

The EDT statement symbol does not need to be specified.

Program execution (initialization, switching to screen dialog, termination with unloading and release of memory) is controlled by the statements passed to EDT.

Once EDT has been loaded, its data area is initialized (only during the first call).

After execution of the statements, EDT returns control to the calling program.

A @HALT statement in the statement sequence causes EDT to be terminated (with release of memory and unloading of EDT).

If there is no @HALT statement at the end of the sequence, control is returned to the calling program without releasing the data area. Processing can now be continued by issuing a new call with a statement sequence, or EDT can be terminated by passing a @HALT statement.

The statement sequence may be up to 256 bytes long. If the statement field is empty (length 4), control is immediately returned to the calling program.

If an error (syntax or runtime error) occurs, execution of the statements is terminated immediately with an appropriate return code and an error message. In the case of a syntax error, field EGLCMDS (EDTGLCB) acts as an error pointer (containing the offset between the start of the statement sequence and the statement which is incorrect). Return code EUPSYERR is returned.

Screen dialog

Switching to screen dialog is possible with the aid of a @DIALOG statement in the statement sequence passed to EDT.

The messages passed across the interface are displayed in the message line each time EDT switches to F mode screen dialog.

@EDIT ONLY switches to line mode dialog (reading from RDATA).

The dialog is terminated by means of a @END, @HALT or @RETURN statement or, for screen dialog, by hitting the K1 key.

EDT passes a return code to the global control block EDTGLCB (EGLRETC). After termination of the screen dialog, execution of the statement sequence is continued. @END sets the same return code as @HALT.

If <message> is specified in a @HALT or @RETURN statement, this message text is also placed in the message field of EDTGLCB.

If the dialog was terminated with @HALT ABNORMAL, the main return code EUPABERR is set.

If the flag EUPNTXT is set in EDTUPCB, the specification of <message> or ABNORMAL is rejected with an error message (in the dialog).

This information can be used by the calling program to control processing.

The flag EGLSTXIT in EDTGLCB is evaluated for each call via the IEDTCMD interface. When a return is made to the calling program the interrupt routines of EDT are exited (if they had been requested).

Setting the flag EUPNUSER in EDTUPCB causes rejection not only of the execution of a @RUN statement but also of a @USE statement in the dialog.

Control structures

The following data areas must be defined in the main program before the CMD function is called:

– the control block EDTGLCB
– the control block EDTUPCB
– the statement sequence (COMMAND)
– 2 message lines (MESSAGE1 and MESSAGE2)

COMMAND - passing a statement sequence

A statement sequence must be passed whenever the CMD function is called.
It is passed in the COMMAND field, which is defined in the main program.

Length :	max. 260 bytes (4+256)
Format :	variable

MESSAGE1, MESSAGE2 - passing a message

Whenever control is transferred from the main program to EDT, two message lines can be passed. These are displayed on the screen when the user switches to screen dialog mode.

MESSAGE1:	1st message line
MESSAGE2:	2nd message line (if the screen is split)

These two data lines must be defined in the main program.

Length:	max. 84 characters (4+80)

Format: variable

If the length field of MESSAGE1 or MESSAGE2 contains a value less than or equal to 4, output of the corresponding message line is suppressed. If this is the case in the first call, the EDT start message is entered.

If @DIALOG is the last statement and MESSAGE1 and MESSAGE2 are specified, these messages are always displayed.

If the @DIALOG statement is not the last statement in the sequence, it should be noted that the return code and the message in EDTGLCB may be overwritten by subsequent statements.

Call

The following information is required (see table below):

– the entry point address IEDTCMD must be specified
– the parameter list with the addresses of the data areas EDTGLCB, EDTUPCB, COMMAND, MESSAGE1 and MESSAGE2 must be set up
– the control block fields in EDTGLCB and EDTUPCB must be filled with the call parameters
– the statement sequence (call parameters) must be placed in the COMMAND field
– message texts must be placed in fields MESSAGE1 and MESSAGE2.

Overview table

Control blocks: see section “EDTGLCB - Global control block” on page 53ff.

Entry point address	:	IEDTCMD

Parameter list	:	A (EDTGLCB, EDTUPCB, COMMAND, MESSAGE1, MESSAGE2)

Call parameters		**Return parameters**	
EDTGLCB:	EGLUNIT EGLVERS EGLINDB [EGLDATA] [EGLENTRY]	EDTGLCB:	EGLRETC [EGLRMSG] EGLCMDS EGLFILE
EDTUPCB:	EUPUNIT EUPVERS EUPINHBT		
COMMAND MESSAGE1 MESSAGE2			

i Each time control is returned, the return code and the name of the current work file (EGLFILE) are placed in the EDTGLCB control block.

Return codes

EGLMRET	**EGLRS1**
EUPRETOK	EUPOK00 EUPOK04 EUPOK08 EUPOK12 EUPOK16 EUPOK20
EUPSYERR	00
EUPRTERR	00
EUPEDERR	00
EUPOSERR	00
EUPUSERR	00
EUPPAERR	EUPPA04 EUPPA08 EUPPA12 EUPPA16
EUPSPERR	00
EUPABERR	EUPOK04 EUPOK08

The fields EGLMRET and EGLRS1 are located in control block EDTGLCB. For the meanings of the return codes see section “EDTGLCB - Global control block” on page 53ff.

Example

```
*****************************************************************
* CMDBSP:  EXAMPLE OF THE EXECUTION OF AN EDT STATEMENT CHAIN   *
*          (PAR SPLIT=OFF,LOWER=ON,SCALE=ON,INDEX=ON;DIALOG)    *
*****************************************************************
*
CMDBSP   START
CMDBSP   AMODE ANY
CMDBSP   RMODE ANY
         BALR  R10,0
         USING *,R10
         LA    R13,SAVEAREA
         LA    R1,CMDPL
         L     R15,=V(IEDTCMD)
         BALR  R14,R15
         TERM  ,
*
* DATA AREA
R1       EQU   1
R10      EQU   10
R13      EQU   13
R14      EQU   14
R15      EQU   15
*
SAVEAREA DS    18F
* - CONTROL BLOCKS (EDTGLCB, EDTUPCB)
         IEDTGLCB C
         IEDTUPCB C
* - STATEMENT SEQUENCE (COMMAND)
CMDDIA   DC    Y(CMDDIAL)
         DC    CL2' '
         DC    C'PAR SPLIT=OFF,LOWER=ON,SCALE=ON,INDEX=ON;DIALOG'
CMDDIAL  EQU   *-CMDDIA
* - MESSAGE LINE (MESSAGE1)
MSG1DIA  DC    Y(MSG1DIAL)
         DC    CL2' '
         DC    C'END DIALOG USING HALT OR <K1>'
MSG1DIAL EQU   *-MSG1DIA
* - MESSAGE LINE (MESSAGE2)
MSG2DIA  DC    Y(MSG2DIAL)
         DC    CL2' '
MSG2DIAL EQU   *-MSG2DIA
* - PARAMETER LIST FOR CMD
CMDPL    DC    A(EDTGLCB)
         DC    A(EDTUPCB)
         DC    A(CMDDIA)
         DC    A(MSG1DIA)
         DC    A(MSG2DIA)
*
         END   CMDBSP
```

Call in the C program

Include files required:

– #include <stdio.h>
– #include "iedglcb.h"
– #include "iedupcb.h"

The EDTGLCB control block is declared and initialized as follows:

```
extern void iedtcmd();

struct iedglcb_mdl iedtglcb=IEDGLCB_INIT;

struct iedupcb_mdl iedtupcb=IEDUPCB_INIT;
```

EDTGLCB is initialized once only, i.e. if you use the IEDTINF function you will automatically initialize the control block for this call. An example of the structure and parameter assignment for the command, message1 and message2 structures in variable record format is provided on page 104. The IEDTCMD function is called at the addresses of these structures:

```
iedtcmd(&iedtglcb; &iedtupcb; &command; &message1; &message2)
```

3.2.3 IEDTEXE - Execute EDT statements without screen dialog

With this call, all user programs connected to EDT can pass statements to EDT.

This function differs from the IEDTCMD function in the following points:

– EDT must already be loaded and initialized.

– No screen dialog can be conducted (@DIALOG and @EDIT are not permitted).

– EDT cannot be terminated or unloaded (@HALT and @RETURN are not permitted).

– No EDT procedures can be started (@INPUT and @DO are not permitted).

This interface is required whenever EDT statements are to be effective for the work files of the calling EDT in an external statement routine (see chapter “External statement routines: @USE” on page 107ff). If the IEDTCMD function is used in an external statement routine, EDT is loaded a second time.

Otherwise, all statements which may be used in both F mode and L mode are permitted. The maximum permissible length of the statement sequence is 256 characters.

If a syntax or runtime error occurs, execution of the statements is terminated immediately with an appropriate return code and an error message. In the case of a syntax error, field EGLCMDS (EDTGLCB) acts as an error pointer (containing the offset from the start of the statement sequence to the statement which is incorrect). Return code EUPSYERR is returned.

If the IEDTEXE function is called from an external statement routine, no further external statements may be entered.

Control structures

The following data areas must be defined in the user routine before the IEDTEXE function is called:

– the control block (EDTGLCB)
– the statement or statement sequence (COMMAND)

COMMAND - passing a statement or statement sequence

A statement or statement sequence must be passed whenever the IEDTEXE function is called. It is passed in the COMMAND field, which is defined in the main program.

Length :	max. 260 characters (256 statement characters + 4 bytes)
Format :	variable

Call

The following information is required (see table below):

- the entry point address IEDTEXE must be specified
- the parameter list with the addresses of the data areas EDTGLCB and COMMAND must be created
- the control block fields in EDTGLCB must be filled with the call parameters
- the statement sequence (call parameters) must be placed in the COMMAND field.

Overview table

Control blocks: see section “EDTGLCB - Global control block” on page 53ff.

Entry point address	:	IEDTEXE

Parameter list	:	A (EDTGLCB, COMMAND)

Call parameters		**Return parameters**	
EDTGLCB:	EGLUNIT	EDTGLCB:	EGLRETC
	EGLVERS		[EGLRMSG]
	[EGLDATA]		EGLCMDS
	[EGLENTRY]		EGLFILE
	[EGLINDB]		EGLUSR1
			EGLUSR2
COMMAND			EGLUSR3

Return codes

EGLMRET	**EGLRS1**
EUPRETOK	00
EUPSYERR	00
EUPRTERR	00
EUPEDERR	00
EUPOSERR	00
EUPUSERR	00
EUPPAERR	EUPPA04
	EUPPA12

The fields EGLMRET and EGLRS1 are located in control block EDTGLCB. The meanings of the return codes are described in section “EDTGLCB - Global control block” on page 53ff.

Call in the C program

Include files required:

– #include <stdio.h>
– #include "iedglcb.h"

The EDTGLCB control block is declared and initialized as follows:

```
extern void iedtexe();
```

In the C program, too, the IEDTEXE function is called at the address of EDTGLCB and with the command to be executed:

```
iedtinf(&iedtglcb; &command)
```

3.3 Logical record access functions

With the aid of the logical record access functions, a user program can access the current work files one record at a time using the EDT record access system.

These functions permit file processing at an elementary level, without using EDT statements.

The following access functions are available for file processing:

Access function	**Entry point address**
Read a record	IEDTGET
Read a marked record	IEDTGTM
Write a record	IEDTPUT
Mark a record	IEDTPTM
Delete the copy buffer or a record range	IEDTDEL
Modify a record index	IEDTREN
Provide information on the work file status	IEDTGET

The various access functions are described in section “IEDTGET - Read a record” on page 31ff.

An access function can be executed only after EDT has been initialized by means of any CMD call (for more information on the CMD function, see section “IEDTCMD - Execute EDT statements” on page 18ff).
If record access functions are used on a work file which is currently being executed as a procedure by means of @DO, they are rejected with the following return code:
EGLMRET contains the value EAMACERR.
EGLSR1 contains the value EAMAC48 (work file is active).

Processing files and library elements

SAM or ISAM files and program libraries can be processed. These files or elements must have been opened and their contents read into a work file (e.g. by means of the CMD function).

For real processing, the file/element must be read into work file 0.

For virtual processing, the file/element may be read into any of the work files 0 to 22.

Access to a specific record is performed via the index, under which a record and a two-byte mark field are stored.

The file or element can be written back to disk only by EDT (e.g. by means of the CMD function).

Control structures

The following data areas must be defined in the calling program before a record access function is called:

- the control block EDTGLCB (see page 53ff)
- the control block EDTAMCB (for record access functions)
- the control blocks EDTPARG and EDTPARL (for work file status)
- the field EDTREC
- the fields EDTKEY, EDTKEY1, EDTKEY2 (for record index)

EDTREC

This field is used

- to pass the record to the main program (return parameter) when reading a record (IEDTGET) or reading a marked record (IEDTGTM)
- to pass the record to EDT (call parameter) when writing a record (IEDTPUT).

Record length:	minimum 1byte
	maximum 256 bytes.

The length of the record must be placed in field EAMLREC of control block EDTAMCB.

EDTKEY1, EDTKEY2, EDTKEY

These fields must always contain a record index in index format. Which of the fields is/are used depends on the access function to be executed.

EDTKEY1 and **EDTKEY2**

These fields are used to pass:

- the index of a single record (e.g. EDTKEY1)
- the indices of a record range (EDTKEY1, EDTKEY2)

EDTKEY1 and EDTKEY2 are always used as input parameters. The lengths of the parameters (8) are placed in fields EAMLKEY1 and EAMLKEY2 in EDTAMCB.

EDTKEY

This contains the index of the record (EDTREC) to be processed or of the mark field (EAMMARK in EDTAMCB).
EDTKEY is used as a call parameter and a return parameter.
The length of EDTKEY in EAMLKEY (EDTAMCB) must be 8.

Index format

The EDT record index is 8 bytes long (fixed). The permissible index range in an EDT work file extends

from ’00000001’	(hexadecimal 'F0F0F0F0F0F0F0F1')
to ’99999999’	(hexadecimal 'F9F9F9F9F9F9F9F9')

The fields EDTKEY1, EDTKEY2 and EDTKEY may contain only values within this range. The associated length fields in EDTAMCB must be set by the user to H’8’. Specifications outside this range will result in an error code.

In order to address the first or last record of a work file with the access functions

– read record (IEDTGET)
– read marked record (IEDTGTM)
– delete record range (IEDTDEL)

the input parameter may also be specified in binary form as follows:

for the first record:	binary zeros (hexadecimal '0000000000000000')
for the last record:	binary ones (hexadecimal 'FFFFFFFFFFFFFFFF')

3.3.1 Call types - transfer modes LOCATE and MOVE

Transfer mode defines how the fields EDTKEY1, EDTKEY2, EDTKEY and EDTREC are to be passed.

Transfer mode is specified by the value in field EAMMODB of control block EDTAMCB:

– LOCATE mode: EAMMODBDC X´04´

– MOVE mode: EAMMODBDC X´00´

LOCATE mode

In LOCATE mode, pointers to the fields (EDTKEY1, EDTKEY2, EDTKEY and EDTREC) must be passed as parameters. EDT modifies these pointers to act as return parameters.

MOVE mode

In MOVE mode, the fields EDTKEY1, EDTKEY2, EDTKEY and EDTREC must be passed as parameters. EDT modifies these fields to act as return parameters. EAMPKEY and EAMPREC contain the buffer lengths of the output parameters.

Overview of the parameter lists for the call types

Mode	**LOCATE mode**	**MOVE mode**
Parameter list	A (EDTGLCB, EDTAMCB, [A(EDTKEY1),] [A(EDTKEY2),] [A(EDTKEY),] [A(EDTREC)])	A (EDTGLCB, EDTAMCB, [EDTKEY1,] [EDTKEY2,] [EDTKEY,] [EDTREC])

The call parameters are contingent on the record access function to be used. The sections describing the various access functions indicate which parameters must be explicitly specified.

3.3.2 IEDTGET - Read a record

This access function can be used to read a record from a file.

The record to be read is defined by the following specifications:

– a work file variable ($0-$22) in field EAMFILE (EDTAMCB).
  This specifies which work file contains the record to be read

– the index of the desired record of the file in field EDTKEY1
  The following specification is also valid:

  for reading the first record of a file: X'0000000000000000'
  for reading the last record of a file : X'FFFFFFFFFFFFFFFF'

– displacement n (0, +N,−N) in field EAMDISP (EDTAMCB), specifying the displacement from the specified index.

Two types of addressing are possible:
– absolute addressing
– relative addressing

1. Reading a record with a specific index - absolute addressing

Specification of the displacement (EAMDISP): n = 0
If the displacement is specified as 0, EDT searches for the record with the specified index (EDTKEY1).
If there is no record with this index, EDT reads and returns the next record (which may be the first or last record).
Overview of return codes and the record actually read (OUTPUT)

EAMFILE KEY1 DISPLACEMENT	: : : 0	File not empty (VARIABLE) (CONSTANT)		
KEY1	EAMDISP	EGLMRET	EAMSR1	OUTPUT
KEY1 = KEY of an existing record	0	EAMRETOK	EAMOK00	Record with KEY = KEY1
KEY1 < KEY of the first record	0	EAMRETOK	EAMOK08	First record
KEY1 >KEY of the last record	0	EAMRETOK	EAMOK12	Last record
KEY1 > KEY of the first record and KEY1 < KEY of the last record and no KEY of an existing record	0	EAMRETOK	EAMOK04	Next record after KEY1

For the meanings of the return codes, see section “EDTGLCB - Global control block” on page 53ff. The above overview applies if the work file which is to be read is not empty.

2. Reading a record with relative addressing

Specification of the displacement (EAMDISP): n = +N/–N (N≠0)

The address of the record to be read is formed from

– the index of a freely selectable record in the file (EDTKEY1)

– the displacement N from the specified index.

+N : the Nth record after the index is read.

–N : the Nth record before the index is read.

If the desired record does not exist, the first or last record, respectively, is returned.

Overview of return codes and record actually read (OUTPUT)

EAMFILE KEY1 DISPLACEMENT	: : : +/− N	File not empty (VARIABLE) (VARIABLE , N≠0)		
KEY 1	EAMDISP	EGLMRET	EGLSR1	OUTPUT
KEY1 = XXXXXXXX	+N (N <= lines after KEY1)	EAMRETOK	EAMOK00	Record N after KEY1
KEY1 = XXXXXXXX	−N (N <= lines before KEY1)	EAMRETOK	EAMOK00	Record N before KEY1
KEY1 XXXXXXXX	+N (N > lines after KEY1)	EAMRETOK	EAMOK12	Last record
KEY1 XXXXXXXX	−N (N > lines before KEY1)	EAMRETOK	EAMOK08	First record

For the meanings of the return codes see section “EDTGLCB - Global control block” on page 53ff.

Records with record mark 13 (see section “IEDTPTM - Mark a record” on page 42) are considered only if the EAMIGN13 flag has been set in the EAMFLAG field of the EDTAMCB control block.

Call

The following information is required (see table below):

– the entry point address IEDTGET must be specified
– the parameter list (dependent on the transfer mode) must be created
– the control block fields and data fields (depending on the transfer mode) must be supplied with values.

Overview table

Control blocks: see section “EDTGLCB - Global control block” on page 53ff.

Entry point address	:	IEDTGET

Parameter list	:	
MOVE mode	:	A (EDTGLCB,EDTAMCB,EDTKEY1, EDTKEY, EDTREC)
LOCATE mode	:	A (EDTGLCB, EDTAMCB, A(EDTKEY1),A(EDTKEY), A(EDTREC))

Call parameters			**Return parameters**		
EDTGLCB:			EDTGLCB:	EGLRETC	
				[EGLRMSG]	(2)
EDTAMCB:	EAMMODB				
	EAMFILE		EDTAMCB:	EAMMARK	
	EAMDISP			EAMLKEY	
	EAMLKEY1			EAMLREC	
	[EAMPKEY]	(1)			
	[EAMPREC]	(1)	EDTKEY		
			EDTREC		
EDTKEY1					
EDTKEY					
EDTREC					

(1) In MOVE mode only
(2) Dependent on the return code

Return parameters after successful record access

In addition to the EGLRETC field in EDTGLCB (EGLMRET = EAMRETOK), EDT returns the following parameters:

Record	EDTREC
Record length	EAMLREC in EDTAMCB
Index	EDTKEY
Index length	EAMLKEY in EDTAMCB
Mark field	EAMMARK in EDTAMCB

If record access was not successful, fields EAMLKEY and EAMLREC contain the value 0.

Call in the C program

Include files required:

– #include <stdio.h>
– #include "iedglcb.h"
– #include "iedamcb.h"

The EDTAMCB control block is declared and initialized as follows:

```
extern void iedtget();

struct iedamcb_mdl iedtamcb=IEDAMCB_INIT;
iedtamcb.mode_flag.mode_byte=0;
MOVE mode:
     char edtrec[256];
     char edtkey, edtkey1[8];
     iedtamcb.length_key_outbuffer=8;
     iedtamcb.length_rec_outbuffer=256;

LOCATE mode:
     iedtamcb.mode_flag.mode_bits.locate=1;
     const char * pedtrec;
     const char * pedtkey, pedtkey1;
iedtamcb.length_key1=iedtamcb.length_key2=iedtamcb.length_key=8;
```

The values for the other parameters are user-dependent. If, for example, the first record in the work file 0 is searched for:

```
char localfile [9] = "    $0";               /* "------$0" */
strncpy(iedtamcb.filename,localfile,8);
char first [9] = "00000001";
strncpy(edtkey1,first,8);
iedtamcb.displacement = 0;
```

In the C program, too, the call for the IEDTGET function differs in the LOCATE and MOVE transfer modes:

MOVE mode	`iedtget(&iedtglcb; &edtamcb; edtkey1; edtkey; edtrec)`
LOCATE mode	`iedtget(&iedtglcb;&edtamcb; &pedtkey1; &pedtkey; &pedtrec)`

3.3.3 IEDTGTM - Read a marked record

This access function makes it possible to search, starting at a specific index (EDTKEY1), for a record with an EDT record mark and to read this record.
The direction for the search, i.e. backward or forward, can also be specified.
The search can be performed in work files 0 through 22. If an ISAM file has been opened for real processing by means of @OPEN, format 1, the access attempt is rejected with a return code.

Searching for a marked record

In order to search for a marked record, the following must be specified:

– a work file variable ($0-$22) in field EAMFILE (EDTAMCB); this specifies the work file containing the marked record.

– the index of a freely selectable record in this file in field EDTKEY1
The following may also be specified here:

  for reading the first record of a file: X'0000000000000000'
  for reading the last record of a file: X'FFFFFFFFFFFFFFFF'

– displacement n (0, +1, –1) in field EAMDISP (EDTAMCB); this specifies the direction of the search.

Two types of search are possible:
– search for a specific index
– search for the next marked record before or after a specified index

1. Reading a marked record with a specific index

Displacement n = 0

If the displacement (EAMDISP) is specified as 0, EDT searches for and reads the record with the specified index (EDTKEY1). If there is no record with this index, or if the record with the index is not marked, the next marked record (which may be the first or last marked record) is read and returned.

Overview of the return codes and the record actually read (EDTREC)

EAMFILE KEY1 DISPLACEMENT	: : : 0	File contains marked records (VARIABLE) (CONSTANT)		
KEY1	EAMDISP	EGLMRET	EGLRS1	EDTREC
KEY1 = KEY of an existing record	0	EAMRETOK	EAMOK00	Record with KEY = KEY1
KEY1 < KEY of the first marked record	0	EAMRETOK	EAMOK08	First record with marking
KEY1 > KEY of the last marked record	0	EAMRETOK	EAMOK12	Last record with marking
KEY > KEY of the first marked record and KEY1 < KEY of the last marked record and not KEY of the marked record	0	EAMRETOK	EAMOKO4	Next marked record after KEY1

The above overview applies if the work file which is to be read contains at least one marked record.
The meanings of the return codes are described in more detail in section “EDTGLCB - Global control block” on page 53ff.

2. Reading the next marked record

Displacement n = +1 or –1

EDT is to search for and read the next marked record before or after a specific index (EDTKEY1).

n = +1:	the next marked record after the specified index is read
n = –1:	the next marked record before the specified index is read

If there are no more marked records in the specified direction, the first or last marked record is read and returned.

Overview of the return codes and the record actually read (EDTREC)

EAMFILE	:	File contains marked records		
KEY1	:	(VARIABLE)		
DISPLACEMENT	: +1 or −1	(VARIABLE , N≠0)		
KEY1	EAMDISP	EGLMRET	EGLRS1	EDTREC
KEY = XXXXXXXX	+1 Record with marking exists after KEY1	EAMRETOK	EAMOK00	Next record with marking after KEY1
KEY1 = XXXXXXXX	−1 Record with marking exists before KEY1	EAMRETOK	EAMOK00	Next record with marking before KEY1
KEY1 = XXX XXXXX	+1 No record with mar-king after KEY1	EAMRETOK	EAMOK12	Last marked record
KEY1 = XXXXXXXX	−1 No record with mar-kingbefore KEY1	EAMRETOK	EAMOK08	First marked record

The above overview applies if the file which is to be read contains at least one marked record.

The meanings of the return codes are described in more detail in section “EDTGLCB - Global control block” on page 53ff.

Call

The following information is required (see table below):

– the entry point address IEDTGTM must be specified
– the parameter list must be created (depending on the transfer mode)
– the control block fields and data fields must be filled (depending on the transfer mode).

Overview table

Control blocks: see section “EDTGLCB - Global control block” on page 53ff.

Entry point address	:	IEDTGTM

Parameter list	:	
MOVE mode	:	A (EDTGLCB,EDTAMCB, EDTKEY1, EDTKEY, EDTREC)
LOCATE mode	:	A (EDTGLCB, EDTAMCB, A(EDTKEY1),A(EDTKEY), A(EDTREC))

Call parameters			Return parameters		
EDTGLCB:			EDTGLCB:	EGLRETC	
				[EGLRMSG]	(2)
EDTAMCB:	EAMMODB				
	EAMFILE		EDTAMCB:	EAMMARK	
	EAMDISP			EAMLKEY	
	EAMLKEY1			EAMLREC	
	[EAMPKEY]	(1)			
	[EAMPREC]	(1)	EDTKEY		
			EDTREC		
EDTKEY1					
EDTKEY					
EDTREC					

(1) In MOVE mode only
(2) Dependent on the return code

Return parameters after successful record access:

In addition to field EGLRET of EDTGLCB (EGLMRET = EAMRETOK), EDT sets the following parameters:

Record	EDTREC
Record length	EAMLREC in EDTAMCB
Index	EDTKEY
Index length	EAMLKEY in EDTAMCB
Mark field	EAMMARK in EDTAMCB

Call in the C program

Include files required:

- #include <stdio.h>
- #include "iedglcb.h"
- #include "iedamcb.h"

The EDTAMCB control block is declared and initialized as follows:

```
extern void iedtgtm();

struct iedamcb_mdl iedtamcb=IEDAMCB_INIT;
iedtamcb.mode_flag.mode_byte=0;
iedtamcb.mode_flag.mode_bits.locate = 1;
const char * pedtrec, pedtkey, pedtkey1;
```

```
char edtkey1 [8];
pedtkey1 = *edtkey1;
iedtamcb.length_key1=iedtamcb.length_key2=iedtamcb.length_key=8;
```

The values for the other parameters are user-dependent. If, for example, the next marked record after the record with the line number 123.4 is searched for in work file 22:

```
char localfile [9] = "     $22";      * "-----$22" */
strncpy(iedtamcb.filename,localfile,8);
char zlnr [] = "01234000";
strncpy(edtkey1,zlnr,8);
iedtamcb.displacement = 1;
```

In the C program the IEDTGTM function is called as follows in LOCATE mode:

```
iedtgtm(&iedtglcb;&edtamcb;&pedtkey1; &pedtkey; &pedtrec)
```

3.3.4 IEDTPUT - Write a record

This access function stores a record (EDTREC) and a mark (EAMMARK in EDTAMCB) under a specified index (EDTKEY) in a work file.

If there is already a record with this index in the file, it and its mark are overwritten (see section “IEDTPTM - Mark a record” on page 42ff).

If, when writing a record with IEDTPUT, the mark EAMNOMOD in the EAMFLAG field in the EDTAMCB control block is set, the work file is **not** marked as modified, even though a record has been added. The “modified” flag of the work file is not changed.

This makes it easier for the calling program to see whether a user has modified the work file in interactive mode (namely by querying the EPLMODF field in the EDTPARL control block). Another advantage is that there is no further need for an EDT save query with the HALT statement if nothing has been changed in the dialog.

Call

The following information is required (see table below):

– the entry point address IEDTPUT must be specified
– the parameter list must be created (depending on the transfer mode)
– the control block fields must be supplied with values
– the record must be specified in the EDTREC field
– the record index must be specified in the EDTKEY field.

Overview table

Control blocks: see section “EDTGLCB - Global control block” on page 53ff.

Entry point address	:	IEDTPUT

Parameter list	:	
MOVE mode	:	A (EDTGLCB,EDTAMCB, EDTKEY, EDTREC)
LOCATE mode	:	A (EDTGLCB, EDTAMCB, A(EDTKEY), A(EDTREC))

Call parameters		**Return parameters**		
EDTGLCB:		EDTGLCB:	EGLRETC [EGLRMSG]	 (1)
EDTAMCB:	EAMMODB EAMFLAG EAMFILE EAMMARK EAMLKEY EAMLREC			
EDTKEY EDTREC				

(1) Dependent on the return code

Call in the C program

Include files required:

– #include <stdio.h>
– #include "iedglcb.h"
– #include "iedamcb.h"

The EDTAMCB control block is declared and initialized as follows:

```
extern void iedtput();
```

```
struct iedamcb_mdl iedtamcb=IEDAMCB_INIT;
IEDAMCB_SET_NO_MARKS(iedtamcb);
iedtamcb.mode_flag.mode_byte=0;
iedtamcb.length_key = 8;
```

If, for example, a record with the line number 75 is to be written into work file 0:

```
char localfile [9] = "      $0";           /* "------$0" */
strncpy(iedtamcb.filename,localfile,8);
char edtrec [] = "Das ist der Inhalt des Satzes";
char edtkey [] = "00750000";
iedtamcb.length_rec = 29;
```

In the C program the IEDTPUT function is called as follows in MOVE mode:

```
iedtput(&iedtglcb; &edtamcb; edtkey; edtrec);
```

3.3.5 IEDTPTM - Mark a record

With this access function, it is possible to mark a record in a work file (see the "EDT Statements" manual [1]).

Records in files opened for real processing by means of @OPEN cannot be marked.

EDT stores a bit mark for this record, logically ORing it with all other bit marks. If all bits of the mark field are zero, the mark is deleted.

Possible record marks

- Record marks 1 to 9 (EAMMK01 to EAMMK09 in EDTAMCB) can be assigned as desired. The user can change these marks using the functions IEDTPUT (write record and marks) and IEDTPTM (write record mark).
  Statements (or statement codes) can also be used to set or delete these marks.
- Record mark 13 (EAMMK13 in EDTAMCB) has the special function of an ignore indicator (see the example on page 122).
  Records marked in this way are:
  - deleted automatically on return to the main program
  - not included when writing to a file or to a library element
  - not copied when copying lines
  - not covered by the record access functions IEDTGET and IEDTPTM unless the EAMIGN13 flag is set in the EAMFLAG field of the EDTAMCB control block.
- Record mark 14 (EAMMK14 in EDTAMCB) has the special function of an update indicator. Records with record mark 14 are displayed as overwritable in F mode, even if the message has only been sent off with $\boxed{\text{DUE}}$.
- Record mark 15 (EAMMK15 in EDTAMCB) has the special function of a write-protection indicator. Records with record mark 15 are write-protected, i.e. they cannot be set to overwritable in the F mode screen dialog by means of statement code X or key $\boxed{\text{F2}}$.

EDT cannot interpret record marks 14 and 15 unless PROTECTION=ON is set by means of the @PAR statement.

Record marks 13, 14 and 15 can be modified only by the record access functions IEDTPUT and IEDTPTM, i.e. they cannot be modified by EDT statements. The only exception to this is format 2 of the @COMPARE statement, which deletes all record marks.

Record marks 14 and 15 are deleted by modifying the record on the screen.

The index of the record to be marked must be placed in the EDTKEY field.

If there is no record with the specified key, or if the file has been opened for real processing by means of @OPEN, the call is rejected with a return code.

Call

The following information is required (see table below):

– the entry point address IEDTPTM must be specified

– the parameter list must be created (depending on the transfer mode)

– the control block fields must be supplied with values

– the record index must be specified in the EDTKEY field.

Overview table

Control blocks: see section “EDTGLCB - Global control block” on page 53ff.

Entry point address	:	IEDTPTM

Parameter list	:	
MOVE mode	:	A (EDTGLCB,EDTAMCB, EDTKEY)
LOCATE mode	:	A (EDTGLCB, EDTAMCB, A(EDTKEY)

Call parameters		**Return parameters**		
EDTGLCB:		EDTGLCB:	EGLRETC [EGLRMSG]	 (1)
EDTAMCB:	EAMMODB EAMFILE EAMMARK EAMLKEY EAMLREC			
EDTKEY				

(1) Dependent on the return code

Call in the C program

Include files required:

– #include <stdio.h>

– #include "iedglcb.h"

– #include "iedamcb.h"

The EDTAMCB control block is declared and initialized as follows:

```
extern void iedtptm();

struct iedamcb_mdl iedtamcb=IEDAMCB_INIT;
IEDAMCB_SET_NO_MARKS(iedtamcb);
iedtamcb.mode_flag.mode_byte=0;
iedtamcb.mode_flag.mode_bits.locate = 1;
const char * pedtkey;
iedtamcb.length_key = 8;
```

If, for example, a record with line number 123.4 is to be displayed as overwritable in work file 1 in F mode (record mark 14):

```
char localfile [9] = "      $1";                             /* "------$1" */
strncpy(iedtamcb.filename,localfile,8);
strcpy (*pedtkey,"01234000");
iedtamcb.marks.mark_bytes.upper_marks.mark2_bits.mark_14 = 1;
```

In the C program the IEDTPUT function is called as follows in LOCATE mode:

```
iedtptm(&iedtglcb; &edtamcb; &pedtkey);
```

3.3.6 IEDTDEL - Delete the copy buffer or a record range

This access function deletes the copy buffer or the specified range of records from an existing work file.
Deleting the copy buffer corresponds to the statement code * in F mode.

Deleting the copy buffer

The value C must be stored in the EAMFILE field of the EDTAMCB control block.
The values must be left-justified. The EAMFILE ffield must be filled with blanks.

Deleting a record range

The record range is specified by means of two index values:

EDTKEY1:	first record in the range
EDTKEY2:	last record in the range

X'0000000000000000' specifies the first record of the work file (%)
X'FFFFFFFFFFFFFFFF' specifies the last record of the work file ($)
If X'0000000000000000' is specified for EDTKEY1 and X'FFFFFFFFFFFFFFFF' is specified for EDTKEY2, then all records in the work file will be deleted.

Call

The following information is required (see table below):

- the entry point address IEDTDEL must be specified
- the parameter list must be created (depending on the transfer mode)
- the control block fields and data fields must be supplied with values
- the index values must be specified in EDTKEY1 and EDTKEY2 (if the copy buffer is deleted, these fields are not evaluated; however, they must still be transferred)
- the work file variable ($0 through $22) must be specified in the EAMFILE field in order to delete a record range, or the value C must be specified in order to delete the copy buffer

Overview table

Control blocks: see section “EDTGLCB - Global control block” on page 53ff.

Entry point address	:	IEDTDEL

Parameter list	:	
MOVE mode	:	A (EDTGLCB,EDTAMCB, EDTKEY1, EDTKEY2)
LOCATE ode	:	A (EDTGLCB, EDTAMCB, A(EDTKEY1), A(EDTKEY2))

Call parameters		**Return parameters**		
EDTGLCB:		EDTGLCB:	EGLRETC	
			[EGLRMSG]	(1)
EDTAMCB:	EAMMODB			
	EAMFILE			
	EAMMARK			
	EAMLKEY			
	EAMLREC			
EDTKEY1				
EDTKEY2				

(1) Dependent on the return code

Call in the C program

Include files required:

– #include <stdio.h>
– #include "iedglcb.h"
– #include "iedamcb.h"

The EDTAMCB control block is declared and initialized as follows:

```
extern void iedtdel();
```

```
struct iedamcb_mdl iedtamcb=IEDAMCB_INIT;
iedtamcb.mode_flag.mode_byte=0;
iedtamcb.mode_flag.mode_bits.locate = 1;
const char * pedtkey1, pedtkey2;
iedtamcb.length_key1 = iedtamcb.length_key2 = iedtamcb.length_key = 8;
iedtamcb.length_rec = 256;
```

If, for example, the records from line number 800.001 to the end of work file 1 are to be deleted:

```
char localfile [9] = "      $1";           /* "------$1" */
strncpy(iedtamcb.filename,localfile,8);
strcpy(*pedtkey1,"08000001");
strcpy(*pedtkey2,"99999999");
```

In the C program the IEDTDEL function is called as follows in LOCATE mode:

```
iedtdel(&iedtglcb; &edtamcb; &pedtkey1; &pedtkey2);
```

3.3.7 IEDTREN - Modify the record index

This access function can be used to modify the index of a record in a work file.

EDTKEY1: the index which is to be modified
EDTKEY2: the new index for this record

If there is already a record with the new index value or with an index value which lies between the two specified index values, the function is not executed.

Call

The following information is required (see table below):

- the entry point address IEDTREN must be specified
- the parameter list must be created (depending on the transfer mode)
- the control block fields must be supplied with values
- the index values must be specified in EDTKEY1 and EDTKEY2.

Overview table

Control blocks: see section "EDTGLCB - Global control block" on page 53ff.

Entry point address	:	IEDTREN

Parameter list	:	
MOVE mode	:	A (EDTGLCB,EDTAMCB, EDTKEY1, EDTKEY2)
LOCATE mode	:	A (EDTGLCB, EDTAMCB, A(EDTKEY1), A(EDTKEY2))

Call parameters		**Return parameters**		
EDTGLCB:		EDTGLCB:	EGLRETC	
			[EGLRMSG]	(1)
EDTAMCB:	EAMMODB			
	EAMFILE			
	EAMLKEY			
	EAMLREC			
EDTKEY1				
EDTKEY2				

(1) Dependent on the return code

Call in the C program

Include files required:

- #include <stdio.h>
- #include "iedglcb.h"
- #include "iedamcb.h"

The control block EDTAMCB is declared and initialized as follows:

```
extern void iedtren();

struct iedamcb_mdl iedtamcb=IEDAMCB_INIT;
iedtamcb.mode_flag.mode_byte=0;
iedtamcb.length_key1 = iedtamcb.length_key2 = iedtamcb.length_key = 8;
iedtamcb.length_rec = 256;
```

If, for example, the record with line number 123.4 is the first record in work file 1 and is to be assigned line number 0.1:

```
char localfile [9] = "      $1";           /* "------$1" */
strncpy(iedtamcb.filename,localfile,8);
char edtkey1 [] = "01234000";
char edtkey2 [] = "00000100";
```

In the C program the IEDTREN function is called as follows in MOVE mode:

```
iedtren(&iedtglcb; &edtamcb; edtkey1; edtkey2);
```

3.3.8 IEDTGET - Read the work file status

The IEDTGET function can be used to read global or work file-specific (local) status information on the work files. One of the following values must be placed in the EAMFILE field in the EDTAMCB control block:

G	global values (EDT statement symbol, window sizes,...)
L0 to L22	local values (1st line in the work window, LOWER ON/OFF,...)

The values must be left-justified. The EAMFILE field must be padded with blanks.

EDT stores the status information in control block EDTPARG or EDTPARL in the relevant fields (for more information see section “EDTPARG/EDTPARL - Global and local parameter settings” on page 66ff).

Call

The following information is required (see table below):

– the entry point address IEDTGET must be specified
– the parameter list must be created
– the control block and data fields must be supplied with values
– the required status information must be specified in the EAMFILE field.

Overview table

Control blocks: see section “EDTGLCB - Global control block” on page 53ff.

Entry point address	:	IEDTGET

Parameter list	:	
MOVE mode	:	A (EDTGLCB,EDTAMCB, EDTKEY1, EDTKEY2, EDTPARG/EDTPARL)

The EDTKEY1 and EDTKEY2 fields are not interpreted, but they must nevertheless be passed (mandatory dummy parameters).

Call parameters		**Return parameters**	
EDTGLCB:	EGKUNIT EGLVERS	EDTGLCB:	EGLRETC EGLRMSG
EDTAMCB:	EAMFILE EAMMODB EAMPREC EAMLREC	EDTAMCB: EDTPARG / EDTPARL	EAMLREC

Return codes for querying the status

EGLMRET	EGLSR1
EAMRETOK	EAMOK00
EAMACERR	EAMAC12
EAMACERR	EAMAC24
EAMACERR	EAMAC48
EAMPAERR	EAMPA04
EAMPAERR	EAMPA08
EAMPAERR	EAMPA12
EAMPAERR	EAMPA32
EAMPAERR	EAMPA36

After a successful call (return code EAMRETOK/EAMOK00), the required status information is stored in the control block EDTPARG (global) or EDTPARL (local).

For the meanings of the return codes see section “EDTGLCB - Global control block” on page 53ff.

If the call was unsuccessful, these fields remain unchanged.

Call in the C program

Include files required:

– #include <stdio.h>
– #include "iedglcb.h"
– #include "iedamcb.h"
– #include "iedparg.h"

The control block EDTAMCB is declared and initialized as follows:

```
extern void iedtget();

struct iedamcb_mdl iedtamcb=IEDAMCB_INIT;
struct iedparg_mdl iedtparg=IEDPARG_INIT;     /*for global query only*/
struct iedparl_mdl iedtparl=IEDPARG_INIT;     /*for global query only*/
char edtkey1, edtkey2[8];
iedtamcb.mode_flag.mode_byte=0;
iedtamcb.length_key1=iedtamcb.length_key2=iedtamcb.length_key=8;
iedtamcb.length_key_outbuffer=8;
iedtamcb.lemgth_rec_outbuffer=sizeof(iedtparg);
```

Read the global work status:

```
char localfile[9]="       G";
strncopy(iedtamcb.filename, localfile, 8);
```

Read the local work status:

```
char localfile[9]="     L1";
strncopy(iedtamcb.filename, localfile, 8);
```

In the C program the call for the IEDTGET function differs depending on whether the call is local or global:

gobal	`iedtget(&iedtglcb; &edtamcb; edtkey1; edtkey2; &iedtparg)`
local	`iedtget(&iedtglcb; &edtamcb; edtkey1; edtkey2; &iedtparl)`

3.3.9 Return codes from record access functions

This table shows which return codes EDT may return to the main program after the various record access functions.

Return code		Function					
EGLMRET	EGLRS1	GET	GTM	PUT	PTM	REN	DEL
EAMRETOK	EAMOK00	x	x	x	x	x	x
	EAMOK04	x	x				
	EAMOK08	x	x				
	EAMOK12	x	x				
	EAMOK16						x
	EAMOK20						x
EAMACERR	EAMAC04			x			
	EAMAC08	x	x				
	EAMAC12	x	x				
	EAMAC16	x	x			x	x
	EAMAC20		x		x	x	
	EAMAC24	x	x	x	x	x	x
	EAMAC28			x	x		
	EAMAC32				x		
	EAMAC36					x	
	EAMAC40					x	
	EAMAC44					x	
	EAMAC48	x	x	x	x	x	x
EAMEDERR		x	x	x	x	x	x
EAMOSERR		x	x	x	x	x	x
EAMUSERR		x	x	x	x	x	x
EAMPAERR	EAMPA04	x	x	x	x	x	x
	EAMPA08	x	x	x	x	x	x
	EAMPA12	x	x	x	x	x	x
	EAMPA16	x	x	x	x	x	x
	EAMPA20	x	x	x	x	x	x
	EAMPA24	x	x	x	x	x	x
	EAMPA28			x			
	EAMPA32	x	x	x	x	x	x
	EAMPA36	x	x	x	x	x	x

The fields EGLMRET and EGLRS1 are located in control block EDTGLCB.
For the meanings of the return codes see section “EDTGLCB - Global control block” on page 53ff.

3.4 Structure and generation of the control blocks

3.4.1 EDTGLCB - Global control block

EDTGLCB is the global control block within all EDT program interfaces. It contains those data fields needed by the user program or EDT at any interface.

These parameters (data) are passed to EDT or to the main program via this control block.

Creating the control block EDTGLCB

The control block EDTGLCB can be generated by means of the Assembler macro IEDTGLCB.

Name	Operation	Operands
`[name]`	`IEDTGLCB`	$[\left\{\begin{matrix}\underline{D}\\C\end{matrix}\right\}]$[,prefix]

name	– symbolic name of the first DS statement if C is specified. – name of the DSECT if D is specified.
<u>D</u>	A dummy section (DSECT) is generated.
C	A memory section with symbolic addresses is generated (no CSECT statement).
prefix	1 character, which is to be used as the first character of each generated field name. If "prefix" is not specified, E is used as the default value (except in the first name EDTGLCB).

Specification of the IEDTGLCB macro generates the control block EDTGLCB in the following form:

```
          IEDTGLCB

1 EDTGLCB  MFPRE DNAME=EDT, MF=D
2 EDTGLCB  DSECT,
2                 *,##### PREFIX=I, MACID= #####
1 *NAME     IDLKG ID=EDT, SECT=&D, VER=166

1 *--------------- EDT UNIT NUMBER, EDTGLCB VERSION NUMBER --------------
1 EGLUNITC EQU   66                   EDT UNIT NUMBER
1 EGLVERSC EQU   1                    EDTGLCB VERSION NUMBER
1 EGLVERSL EQU   12                   VERSION-LENGTH (INFO)
1 EGLMSGM  EQU   80                   MAX LENGTH FOR MESS

1 *--------------- EDT MAIN RETURN CODES --------------------------------
1 *                    *------ EDT CALL --------------------------------
1 EUPRETOK EQU   X'0000'              NO ERROR
1 EUPSYERR EQU   X'0008'              SYNTAX ERROR IN COMMAND
1 EUPRTERR EQU   X'000C'              RUNTIME ERROR IN COMMAND
1 EUPEDERR EQU   X'0010'              UNRECOVERABLE EDT ERROR
1 EUPOSERR EQU   X'0014'              UNRECOVERABLE SYSTEM ERROR
1 EUPUSERR EQU   X'0018'              UNRECOVERABLE USER ERROR
1 EUPPAERR EQU   X'0020'              PARAMETER ERROR
1 EUPSPERR EQU   X'0024'              REQM ERROR
1 EUPVEERR EQU   X'0028'              VERSION ERROR                V16.5
1 EUPABERR EQU   X'002C'              ABNORMAL HALT BY USER        V16.5
1 *                    *------ EDT ACCESS METHOD -----------------------
1 EAMRETOK EQU   X'0000'              NO ERROR
1 EAMACERR EQU   X'0004'              ACCESS ERROR
1 EAMEDERR EQU   X'0010'              UNRECOVERABLE EDT ERROR
1 EAMOSERR EQU   X'0014'              UNRECOVERABLE SYSTEM ERROR
1 EAMUSERR EQU   X'0018'              UNRECOVERABLE USER ERROR
1 EAMPAERR EQU   X'0020'              PARAMETER ERROR
1 EAMSPERR EQU   X'0024'              REQM ERROR

1 *--------------- EDT SUBRETURN CODE1 ----------------------------------
1 *                    *------ MAIN: EUPRETOK --------------------------
1 EUPOK00  EQU   X'00'                 NO ERROR
1 EUPOK04  EQU   X'04'                 HALT
1 EUPOK08  EQU   X'08'                 HALT <TEXT>
1 EUPOK12  EQU   X'0C'                 RETURN
1 EUPOK16  EQU   X'10'                 RETURN <TEXT>
1 EUPOK20  EQU   X'14'                 K1 KEY
1 EUPOK24  EQU   X'18'                 IGNORE COMMAND
1 *                    *------ MAIN: EUPPAERR --------------------------
1 EUPPA04  EQU   X'04'                 ERROR IN EDTGLCB
```

```
1 EUPPA08  EQU   X'08'                ERROR IN EDTUPCB
1 EUPPA12  EQU   X'0C'                ERROR IN COMMAND PARAMETER
1 EUPPA16  EQU   X'10'                ERROR IN MESSAGE PARAMETER
1 *                     *------ MAIN: EUPPAERR --------------------------
1 EUPVE00  EQU   X'00'                STANDARD VERSION RETURNED
1 EUPVE04  EQU   X'04'                NO VERSION RETURNED
1 *                     *------ MAIN: EAMRETOK --------------------------
1 EAMOK00  EQU   X'00'                NO ERROR
1 EAMOK04  EQU   X'04'                NEXT RECORD RETURNED
1 EAMOK08  EQU   X'08'                FIRST RECORD RETURNED
1 EAMOK12  EQU   X'0C'                LAST RECORD RETURNED
1 EAMOK16  EQU   X'10'                FILE CLEARED
1 EAMOK20  EQU   X'14'                COPY BUFFER CLEARED          V16.6
1 *                     *------ MAIN: EAMACERR --------------------------
1 EAMAC04  EQU   X'04'                PUT RECORD TRUNCATED
1 EAMAC08  EQU   X'08'                KEY TRUNCATED ( MOVE MODE )
1 EAMAC12  EQU   X'0C'                RECORD TRUNCATED (MOVE MODE )
1 EAMAC16  EQU   X'10'                FILE IS EMPTY
1 EAMAC20  EQU   X'14'                NO MARKS IN FILE
1 EAMAC24  EQU   X'18'                FILE NOT OPENED
1 EAMAC28  EQU   X'1C'                FILE REAL OPENED (NO MARKS)
1 EAMAC32  EQU   X'20'                PTM NOT FOUND
1 EAMAC36  EQU   X'24'                REN KEY ERROR
1 EAMAC40  EQU   X'28'                MAX LINE ERROR
1 EAMAC44  EQU   X'2C'                RENUMBER INHIBITED
1 EAMAC48  EQU   X'30'                FILE IS ACTIVE
1 *                     *------ MAIN: EAMPAERR --------------------------
1 EAMPA04  EQU   X'04'                ERROR IN EDTGLCB
1 EAMPA08  EQU   X'08'                ERROR IN EDTAMCB
1 EAMPA12  EQU   X'0C'                FILENAME ERROR
1 EAMPA16  EQU   X'10'                ACCESS FUNCTION ERROR
1 EAMPA20  EQU   X'14'                KEY FORMAT ERROR
1 EAMPA24  EQU   X'18'                KEY LENGTH ERROR
1 EAMPA28  EQU   X'1C'                RECORD LENGTH ERROR
1 EAMPA32  EQU   X'20'                WRONG TRANSFER MODUS BYTE
1 EAMPA36  EQU   X'24'                WRONG VERSION OR UNIT NUMBER

1 *--------------- CONTROL BLOCK EDTGLCB --------------------------------
1 *                       *---- CONTROL BLOCK HEADER --------------------
1 EGLFHE   DS    0XL8                GENERAL OPERAND LIST HEADER
1 EGLIFID  DS    0A                  INTERFACE IDENTIFIER
1 EGLUNIT  DC    AL2(EGLUNITC)       UNIT NUMBER
1          DS    AL1                 RESERVED
1 EGLVERS  DC    AL1(EGLVERSC)       FUNCTION INTERFACE VERSION NUMBER
1 *                        *--- RETURN CODE -----------------------------
1 EGLRETC  DS    0A                  GENERAL RETURN CODE
1 EGLSRET  DS    0AL2                SUBRETURN CODE
1 EGLSR2   DC    AL1(0)              SUBRETURN CODE2
```

```
1 EGLSR1   DC    AL1(0)             SUBRETURN CODE1
1 EGLMRET  DC    AL2(0)             MAIN RETURN CODE
1 *                       *---- RETURN MESSAGE FIELD --------------------
1 EGLINFM  DS    OF                 INFORMATION OF MEMORY SIZE
1 EGLCMDS  DC    F'0'               DISPLACEMENT OF INVALID COMMAND
1 EGLRMSG  DS    0CL82              EDT RETURN MESSAGE
1 EGLRMSGL DC    H'0'               MESSAGE LENGTH
1 EGLRMSGF DC    CL80' '            MESSAGE FIELD
1 *                       *---- EDT GLOBAL PARAMETERS -------------------
1 EGLCDS   DC    X'00'              CODE OF SENDING KEY            V16.4
1 EGLDUE   EQU   X'66'              DUE
1 EGLF1    EQU   X'5B'              F1
1 EGLF2    EQU   X'5C'              F2
1 EGLF3    EQU   X'5D'              F3
1 EGLK1    EQU   X'53'              K1
1 EGLINDB  DC    X'00'              INDICATOR BYTE
1 EGLREOR  EQU   X'20'              REORGANISATION ALLOWED         V16.4
1 EGLSPL   EQU   X'10'              EDT CALL FROM SPL
1 EGLSTXIT EQU   X'08'              EDT STXIT ALLOWED              V16.4
1 EGLINIT  EQU   X'04'              EDT DATA INITIATED
1 EGLDTVD  EQU   X'02'              EDT DATA ADDRESS VALID
1 EGLETVD  EQU   X'01'              EDT ENTRY ADDRESS VALID
1 EGLENTRY DC    A(0)               EDT ENTRY ADDRESS
1 EGLDATA  DC    A(0)               EDT DATA ADDRESS
1 *                       *---- ACTIVE EDT-FILE (OUTPUT)-----------------
1 EGLFILE  DC    CL8' '             INTERN FILE NAME
1 *                       *---- USER PARAMETERS -------------------------
1 EGLUSR1  DC    XL4'00000000'      USER PARAMETER1 (EDT CALLER    )
1 EGLUSR2  DC    XL4'00000000'      USER PARAMETER2 (SUBROUTINE   )
1 EGLUSR3  DC    XL4'00000000'      USER PARAMETER3 (EXIT ROUTINE  )
1 *--------------- LENGTH OF CONTROL BLOCK ------------------------------
1 EGLGLCBL EQU   *-EDTGLCB
```

If the dialog was terminated with @HALT ABNORMAL, the main return code EUPABERR is set.

Meanings of the control block fields		**Length (bytes)**	**Parameter type**	
			Call	**Return**
EGLFHE	Data area of the control block header	(8)*		
EGLIFID	The input fields of the header contain numbers and have the following names:	(4)*	C	
EGLUNIT	Unique identification of EDT (identical at all EDT interfaces)	2	C	
EGLVERS	Control block version number	1	C	
EGLRETC	The output fields of the standard header have the following names and meanings:	(4)*		R
EGLSRET	Area for subcode fields of the return code	(2)*		
EGLSR1	- SUBCODE1: subvalue of the return code, unique within the main value	1		R
EGLSR2	- SUBCODE2: subvalue of the return code, unique within the main value	1		R
EGLMRET	MAINCODE: main value of the return code; these codes are explained in more detail on page 11	2		R
EGLINFM	EDT places the number of memory pages needed for the static data area in this field; see INFO function IEDTINF on page 16ff.	(4)*		R
EGLCMDS	In the case of a syntax error, this field contains the offset of the invalid statement from the start of the statement sequence (CMD function)	4		R
EGLRMSG	Area for passing a message text	(82)*	C	R
EGLRMSGL	Record length field for field EGLRMSG	2	C	R
EGLRMSGF	EDT passes an (error) message to the user program in this field. If no message is passed, the length field contains 0.	80	C	R
EGLCDS	In this field, EDT passes the send key with which the statement was transmitted in F mode dialog.	1		R
EGLDUE	X'66' DUE key			
EGLF1	X'5B' Function key F1			
EGLF2	X'5C' Function key F2			
EGLF3	X'5D' Function key F3			
EGLK1	X'53' Function key K1			
EGLINDB	This indicator byte contains flags with various meanings.	1		

Meanings of the control block fields		**Length (bytes)**	**Parameter type**	
			Call	**Return**
Flag EGLREOR	This flag must be set if EDT data area reorganization is to be allowed.		C	
Flag EGLSPL	This flag must be set if EDT is called from the SPL environment. In all other cases, this flag must be reset.		C	
Flag EGLSTXIT	This flag must be set if the EDT interrupt routines are desired.			
Flag EGLINIT	This flag is set by EDT after the data area has been initialized.		L	R
EGLDATA (flag EGDTVD)	After memory has been requested for the static data area, the address of this area is placed in EGLDATA and the corresponding flag is set. This should not be changed by the user.			
EGLENTRY (flag EGLETVD)	When EDT is called for the first time, the entry address is saved in this field and the corresponding bit is set. This field must not be modified by the calling program; however, the user must reset bit EGLETVD before EDT is called for the first time.		L	R
EGLFILE	EDT places the number of the current work file ($0 to $22) in this field (left-justified). The remainder of the field is filled with blanks. Before an access to the work files, the name of the current work file can be taken from this field.	8	L	R
EGLUSR1, EGLUSR2, EGLUSR3	EDT provides the user program with 3 transfer fields of 4 bytes each: EGLUSR1 is allocated to the calling program. EGLUSR2 is allocated to an external statement routine. EGLUSR3 is reserved for a future interface. Each program can write into "its" field and can read the contents of all 3 fields. (EDT accepts only the field allocated to it at the interface.) The fields are not changed by EDT; they are simply passed on at all program interfaces.	4 4 4	C	R

* Redefined fields

C	Call parameters:	Must be provided by the user before the call.
R	Return parameters:	Are returned by the called program (EDT).
L	Data field:	Must be overwritten with binary zeros before the first call.

3.4.2 EDTUPCB - Subroutine control block

EDTUPCB (subroutine control block) contains the parameters which define the default values of EDT for the CMD function.

The values set in this manner are effective only in screen dialog mode.

Creating the control block EDTUPCB

EDTTUPCB can be generated by means of the Assembler macro IEDTUPCB.

Name	Operation	Operands
`[name]`	`IEDTUPCB`	$[\left\{\begin{matrix}\underline{D}\\C\end{matrix}\right\}]$[,prefix] [,VERSION = 2]

name	– symbolic name of the first DS statement if C is specified. – name of the DSECT if D is specified.
<u>D</u>	A dummy section (DSECT) is generated.
C	A memory section with symbolic addresses is generated (no CSECT statement).
prefix	1 character, which is to be used as the first character of each generated field name. If "prefix" is not specified, E is used by default (except in the first name EDTUPCB).
VERSION	Interface version. Only version 2 of the interface is supported.

Specification of the IEDTUPCB macro generates the control block EDTUPCB in the following form:

```
           IEDTUPCB

1 EDTUPCB  MFPRE DNAME=EDT, MF=D
2 EDTUPCB  DSECT,
2                 *,##### PREFIX=I, MACID= #####
1 *NAME     IDLKG ID=EDT, SECT=&D, VER=166

1 *--------------- EDT UNIT NUMBER, EDTUPCB VERSION NUMBER ---------------
1 EUPUNITC EQU   66                   EDT UNIT NUMBER
1 EUPVERSC EQU   2                    EDTUP VERSION NUMBER
1 EUPCMDM  EQU   (256+4)              EDT COMMAND MAXLENGTH
1 EUPMSGM  EQU   (80+4)               EDT MESSAGE MAXLENGTH

1 *--------------- CONTROL BLOCK EDTUPCB ---------------------------------

1 *                               *----- CONTROL BLOCK HEADER -------------
1 EUPFHE   DS    0XL8                 GENERAL OPERAND LIST HEADER
1 EUPIFID  DS    0A                   INTERFACE IDENTIFIER
1 EUPUNIT  DS    AL2(EUPUNITC)        UNIT NUMBER
1          DS    AL1                  RESERVED
1 EUPVERS  DS    AL1(EUPVERSC)        FUNCTION INTERFACE VERSION NUMBER
1          DS    A                    RESERVED

1 *                               *----- INHIBIT FLAG ---------------------
1 EUPINHBT DC    X'00'                INHIBIT FLAG BYTE

1 EUPNINHB EQU   X'00'                * NO RESTRICTIONS
1 EUPNEXEC EQU   X'01'                * NO MEXEC/MLOAD (@EXEC/@LOAD)
1 EUPNCMDM EQU   X'02'                * NO CMD (@SYSTEM <STRING>)
1 EUPNBKPT EQU   X'04'                * NO BKPT (@SYSTEM)
1 EUPNUSER EQU   X'08'                * NO USER-PROG. (@RUN/@USE)
1 EUPN@EDT EQU   X'10'                * NO @EDIT      (L-MODE : WRTRD)
1 EUPN@EDO EQU   X'20'                * NO @EDIT ONLY (L-MODE : RDATA)
1 EUPNTXT  EQU   X'40'                * NO <TEXT>     (HALT / RETURN)

1          DS    AL3                  RESERVED
1 *-------------------------- LENGTH OF CONTROL BLOCK --------------------
1 EUPUPCBL EQU   *-EDTUPCB
```

Meanings of the control block fields		Length (bytes)	Call parameter
EUPFHE	Data area of the control block header	(8)*	
EUPIFID	The input fields of the header contain numbers with the following meanings:	(4)*	
EUPUNIT	Unique identification of the function unit EDT (identical at all EDT interfaces)	2	C
EUPVERS	Control block version number	1	C
EUPINHBT	By setting the individual bits, certain statements (@EXEC, @LOAD, @USE, @SYSTEM, @RUN) can be disabled for the user in order to prevent undefined termination of EDT.	1	C

* Redefined fields

Meanings of the flags for disabling statements

EUPN@EDT	Disables @EDIT (switch to L mode dialog and read using WRTRD).
EUPN@EDO	Disables @EDIT ONLY (switch to L mode dialog and read using RDATA).
EUPNTXT	Prevents <message> from being specified in @HALT and @RETURN.
EUPNUSER	Disables statements that define or call user routines (@RUN, @USE).

3.4.3 EDTAMCB - Access method control block

EDTAMCB (Access Method Control Block) is the control block for the logical record access functions. It contains the data fields which are needed for an access to the work files.

The user must supply these parameters (call parameters). They are passed to EDT in EDTAMCB.

Creating the control block EDTAMCB

The control block EDTAMCB can be generated by means of the Assembler macro IEDTAMCB.

Name	Operation	Operands
[name]	IEDTAMCB	$[\left\{\begin{array}{c}\underline{D}\\ C\end{array}\right\}]$[,prefix]

name	– symbolic name of the first DS statement if C is specified. – name of the DSECT if D is specified.
<u>D</u>	A dummy section (DSECT) is generated.
C	A memory section with symbolic addresses is generated (no CSECT statement).
prefix	1 character, which is to be used as the first character of each generated field name. If "prefix" is not specified, E is used by default (except in the first name EDTAMCB).

Specification of the IEDTAMCB macro generates the control block EDTAMCB in the following form:

```
           IEDTAMCB

1 EDTAMCB  MFPRE DNAME=EDT, MF=D
2 EDTAMCB  DSECT,
2                 *,##### PREFIX=I, MACID= #####
1 *NAME     IDLKG ID=EDT, SECT=&D, VER=166

1 *--------------- EDT UNIT NUMBER, EDTAMCB VERSION NUMBER ----------------
1 EAMUNITC EQU  66                   EDT UNIT NUMBER
1 EAMVERSC EQU  1                    INTERFACE IDENTIFIER
```

```
1 *--------------- CONTROL BLOCK EDTAMCB ----------------------------------
1 *                       *---CONTROL BLOCK HEADER ------------------------
1 EAMFHE   DS   0XL8                GENERAL OPERAND LIST HEADER
1 EAMIFID  DS   0A                  INTERFACE IDENTIFIER
1 EAMUNIT  DS   AL2(EAMUNITC)       UNIT NUMBER
1          DS   AL1                 RESERVED
1 EAMVERS  DS   AL1(EAMVERSC)       FUNCTION INTERFACE VERSION NUMBER
1          DS   A                   RESERVED
1 *                       *---TRANSFER MODE BYTE -------------------------
1 EAMMODB  DS   X'00'               MODE FLAG
1 EAMMOVM  EQU  X'00'               MOVE MODE FOR RECORD AND KEYS
1 EAMLOCM  EQU  X'04'               LOCATE MODE FOR RECORD AND KEYS
1 *                       *---FLAG-BYTE ------------------------------V16.4
1 EAMFLAG  DC   X'00'               FLAG                              V16.4
1 EAMIGN13 EQU  X'01'               IGNORE LINE MARK 13               V16.4
1 EAMNOMOD EQU  X'02'               INHIBIT SETTING MODIFIED FLAG     V16.6
1          DS   AL2                 RESERVED
1 *                       *---INPUT PARAMETERS ---------------------------
1 EAMFILE  DC   CL8' '              FILE NAME
1 EAMDISP  DC   F'0'                DISPLACEMENT
1 EAMLKEY1 DC   H'0'                LENGTH OF KEY1
1 EAMLKEY2 DC   H'0'                LENGTH OF KEY2
1 *                       *---INPUT PARAMETERS (ONLY IN MOVE MODE)---------
1 EAMPKEY  DC   H'0'                LENGTH OF KEY OUTPUT BUFFER
1 EAMPREC  DC   H'0'                LENGTH OF RECORD OUTPUT BUFFER
1 *                       *---INPUT/OUTPUT PARAMETERS ---------------------
1 EAMLKEY  DC   H'0'                LENGTH OF KEY
1 EAMLREC  DC   H'0'                LENGTH OF RECORD
1 EAMMARK  DS   0H                  LINE MARKS (16 BITS)
1 EAMMARK2 DC   X'00'               UPPER MARKS (8 BITS)
1 EAMMK15  EQU  X'80'               MARK 15 (BIT 2**15)
1 EAMMK14  EQU  X'40'               MARK 14 (BIT 2**14)
1 EAMMK13  EQU  X'20'               MARK 13 (BIT 2**13)
1 EAMMK12  EQU  X'10'               MARK 12 (BIT 2**12)
1 EAMMK11  EQU  X'08'               MARK 11 (BIT 2**11)
1 EAMMK10  EQU  X'04'               MARK 10 (BIT 2**10)
1 EAMMK09  EQU  X'02'               MARK 09 (BIT 2**09)
1 EAMMK08  EQU  X'01'               MARK 08 (BIT 2**08)
1 EAMMARK1 DC   X'00'               LOWER MARKS (8 BIT)
1 EAMMK07  EQU  X'80'               MARK 07 (BIT 2**07)
1 EAMMK06  EQU  X'40'               MARK 06 (BIT 2**06)
1 EAMMK05  EQU  X'20'               MARK 05 (BIT 2**05)
1 EAMMK04  EQU  X'10'               MARK 04 (BIT 2**04)
1 EAMMK03  EQU  X'08'               MARK 03 (BIT 2**03)
1 EAMMK02  EQU  X'04'               MARK 02 (BIT 2**02)
1 EAMMK01  EQU  X'02'               MARK 01 (BIT 2**01)
```

```
1 EAMMK00  EQU  X'01'             MARK 00 (BIT 2**00)
1
1          DS   AL2               RESERVED
1 *---------------------- LENGTH OF CONTROL BLOCK --------------------
1 EAMAMCBL EQU  *-EDTAMCB
```

Meanings of the control block fields		**Length (bytes)**	**Parameter type**	
			Call	**Return**
EAMFHE	Data area of the control block header	(8)*		
EAMIFID	The input fields of the header (EAMIFID) contain numbers and have the following names:	(4)*		
EAMUNIT	Unique identification of the function unit EDT (identical at all EDT interfaces)	2		
EAMVERS	Control block version number	1		
EAMMODB	Mode byte in which the calling program specifies the desired transfer mode, see page 30	1	C	
EAMFLAG Flag EAMIGN13 Flag EAMNOMOD	The byte contains flags for record processing. This flag must be set if records with mark 13 are to be read (IEDTGET) or if records are to be assigned mark 13 (IEDTPTM). This flag must be set if the work file is not to be marked as modified after a record has been written (IEDTPUT).	1	C	
EAMFILE	The work file variable ($0...$22) specifying which work file (0...22) is to be accessed, or G or L0 to L22 for reading the work file status, or C for deleting the copy buffer.	8	C	
EAMDISP	This field contains: the relative position of the desired record for the function "read a record" (IEDTGET) the desired search direction for the function "read a marked record" (IEDTGTM)	4	C	
EAMLKEY1	Record length field for field EDTKEY1	2	C	
EAMLKEY2	Record length field for field EDTKEY2	2	C	
EAMPEY, EAMPREC	In MOVE mode, the lengths of the output buffers for the record index and the record must be placed in these fields before a record is read (IEDTGET, IEDTGTM)	2 2	C C	
EAMLKEY	Record length field for field EDTKEY	2	C	R

Meanings of the control block fields		**Length (bytes)**	**Parameter type**	
			Call	**Return**
EAMLREC	Record length field for field EDTREC	2	C	R
EAMMARK	The record mark(s). This field is used for both input and output. The mark field is managed as additional information for each record. The user may use the marks 01...09 (EAMMK01...EAMMK09) as desired and may also use marks 13, 14 and 15 (EAMMK13, EAMMK14 and EAMMK15) with IEDTPUT and IEDTPTM. All other marks are reserved for special functions	2	C	R

*Redefined field

C	Call parameters:	Must be supplied by the user before the call.
R	Return parameters:	Are returned by the called program (EDT).

3.4.4 EDTPARG/EDTPARL - Global and local parameter settings

When the file status is read using the function IEDTGET, EDT places the information in one of the following control blocks:

– EDTPARG (for global values) or

– EDTPARL (for local values)

Creating the control block EDTPARG

The control block EDTPARG can be generated by means of the Assembler macro IEDTPARG.

Name	Operation	Operands
`[name]`	`IEDTPARG`	$[\left\{\begin{matrix}\underline{\text{D}}\\ \text{C}\end{matrix}\right\}]$[,prefix]

name	– symbolic name of the first DS statement if C is specified. – name of the DSECT if D is specified.
D	A dummy section (DSECT) is generated.
C	A memory section with symbolic addresses is generated (no CSECT statement).
prefix	1 character, which is to be used as the first character of each generated field name. If "prefix" is not specified, E is used by default (except in the first name EDTPARG).

Specification of the IEDTPARG macro generates the control block EDTPARG in the following form:

```
          IEDTPARG

1 EDTPARG  MFPRE DNAME=EDT, MF=D
2 EDTPARG  DSECT,
2                 *,##### PREFIX=I, MACID= #####
1 *NAME     IDLKG ID=EDT, SECT=&D, VER=166

1 *--------------- EDT UNIT NUMBER, EDTPARL VERSION NUMBER --------------
1 EPGUNITC EQU   66                  EDT UNIT NUMBER
1 EPGVERSC EQU   1                   EDTPARG VERSION NUMBER
1 *--------------- CONTROL BLOCK EDTPARL --------------------------------
1 *                       *---- CONTROL BLOCK HEADER --------------------
1 EPGFHE   DS    0XL8                GENERAL OPERAND LIST HEADER
1 EPGIFID  DS    0A                  INTERFACE IDENTIFIER
1 EPGUNIT  DC    AL2(EPGUNITC)       UNIT NUMBER
1          DS    AL1                 RESERVED
1 EPGVERS  DC    AL1(EPGVERSC)       FUNCTION INTERFACE VERSION NUMBER
1          DS    A                   RESERVED
1 *                       *---- OUTPUT FIELDS ---------------------------
1 EPGMODE  DS    CL1                 EDT MODE (F/L/C)
1 EPG@SYM  DS    CL1                 EDT STATEMENT SYMBOL
1 EPGWDS1  DS    H                   SIZE OF WINDOW 1
1 EPGWDS2  DS    H                   SIZE OF WINDOW 2
1 EPGFILE1 DS    CL8                 WORKFILE IN WINDOW 1
1 EPGFILE2 DS    CL8                 WORKFILE IN WINDOW 2

1 *--------------- MINIMUM LENGTH OF CONTROL BLOCK -----------------V16.4
1 EPGPMINL EQU   *-EDTPARG                                         V16.4

1 *--------------- EXTENSION ---------------------------------------V16.4
1 EPGCCSN  DS    CL8                 CODED CHARACTER SET NAME      V16.4

1 *--------------- TOTAL LENGTH OF CONTROL BLOCK ------------------------
1 EPGPARGL EQU   *-EDTPARG
```

Meanings of the control block fields		**Length (bytes)**	**Parameter type**	
			Call	**Return**
EPGFHE	Header	(8)*		R
EPGIFID	Header input fields	(4)*		R
EPGUNIT	Identification of EDT (66)	2	C	
EPGVERS	Update status of the control block	1	C	

EPGMODE	Current mode:
	F = F mode
	L = L mode
	C = Caller
		1		R

EPG@SYM	EDT escape symbol	1		R
EPGWDS1	Size of window 1 (2..24)	2		R
EPGWDS2	Size of window 2 (0..22)	2		R
EPGFIL1	Work file in window 1 ($0..$9)	8		R
EPGFIL2	Work file in window 2 ($0..$9)	8		R
EPGPMINL	Minimum length of control block. If a shorter output buffer than EPGLMINL is defined in field EAMPREC of control block EDTAMCB, the IEDTGET function for reading the global status information is rejected with return code EAMPA08 (error in EDTAMCB).			
EPGCCSN	Coded Character Set Name. Only provided if the receive field length (EAMPREC in EDTAMCB) suffices.	8		R

* Redefined fields

EDT places printable values in all fields which are declared as character data ("C").

Halfword fields ("H") are filled with binary numbers.

Creating the control block EDTPARL

The control block EDTPARL can be generated by means of the Assembler macro IEDTPARL.

Name	Operation	Operands
`[name]`	`IEDTPARL`	$[\left\{\begin{matrix}\underline{D}\\C\end{matrix}\right\}]$[,prefix] [VERSION=$\left\{\begin{matrix}\underline{1}\\2\\3\end{matrix}\right\}$]

name	– symbolic name of the first DS statement if C is specified. – name of the DSECT if D is specified.
<u>D</u>	A dummy section (DSECT) is generated.
C	A memory section with symbolic addresses is generated (no CSECT statement).
prefix	1 character, which is to be used as the first character of each generated field name. If "prefix" is not specified, E is used by default (except in the first name EDTPARL).
VERSION	Version of the interface. Unless otherwise specified, version 1 of the interface is generated for reasons of compatibility. When VERSION=2 is specified, the expanded format with additional information is generated. The following description refers to Version 2 of the interface.
=<u>1</u>	Version 1 of the interface (default value).
=2	Version 2 of the interface.
=3	Version 3 of the interface with additional information on a POSIX file opened with @XOPEN and on the CODE default.

Specification of the IEDTPARL macro generates the control block EDTPARL in the following form (VERSION=2):

```
           IEDTPARL D,VERSION=2

1 EDTPARL  MFPRE DNAME=EDT, MF=D, PREFIX=*
2 EDTPARL  DSECT,
2                 *,##### PREFIX=I, MACID= #####
1 *NAME     IDLKG ID=EDT, SECT=&D, VER=166

1 *-------------------------EDT UNIT NUMBER, EDTPARL VERSION NUMBER-------
1 EPLUNITC EQU   66                  EDT UNIT NUMBER
1 EPLVERSC EQU   3                   EDTPARL VERSION NUMBER
1 *                        *---- CONTROL BLOCK HEADER --------------------
1 EPLFHE   DS    0XL8                GENERAL OPERAND LIST HEADER
1 EPLIFID  DS    0A                  INTERFACE IDENTIFIER
1 EPLUNIT  DC    AL2(EPLUNITC)       UNIT NUMBER
1          DS    AL1                 RESERVED
1 EPLVERS  DC    AL1(EPLVERSC)       FUNCTION INTERFACE VERSION NUMBER
1          DS    A                   RESERVED
1 *                        *---- OUTPUT FIELDS ---------------------------
1 EPLVPOS  DS    CL8                 FIRST LINE IN WINDOW
1 EPLHPOS  DS    H                   FIRST COLUMN IN WINDOW
1 EPLRLIM  DS    H                   MAX RECORD-LENGTH IN F-MODE
1 EPLINF   DS    CL1                 INF ON/OFF (1/0)
1 EPLLOW   DS    CL1                 LOWER ON/OFF (1/0)
1 EPLHEX   DS    CL1                 HEX ON/OFF   (1/0)
1 EPLEDL   DS    CL1                 EDIT LONG ON/OFF (1/0)
1 EPLSCALE DS    CL1                 SCALE ON/OFF (1/0)
1 EPLPROT  DS    CL1                 PROTECTION ON/OFF (1/0)
1 EPLSTRUC DS    CL1                 STRUCTURE SYMBOL
1 EPLOPEN  DS    CL1                 OPEN FLAG:   (R/P/I/S/0)
1 EPLEMPTY DS    CL1                 EMPTY FLAG
1 EPLMODIF DS    CL1                 MODIFIED FLAG
1 EPLSTDF  DS    CL54                STANDARD FILENAME
1 EPLSTDL  DS    CL54                STANDARD LIBRARY NAME
1 EPLSTDT  DS    CL8                 STANDARD PLAM TYPE
1 EPLOPNFL DS    CL54                NAME OF OPENED FILE/PLAM LIBRARY
1 EPLOPNE  DS    CL64                NAME OF OPENED PLAM ELEMENT
1 EPLOPNV  DS    CL24                VERSION OF OPENED PLAM ELEMENT
1 EPLOPNT  DS    CL8                 TYP OF OPENED PLAM ELEMENT
1 EPLVPOS1 DS    CL8                 FIRST LINE IN WINDOW 1
1 EPLHPOS1 DS    H                   FIRST COLUMN IN WINDOW 1
1 EPLVPOS2 DS    CL8                 FIRST LINE IN WINDOW 2
1 EPLHPOS2 DS    H                   FIRST COLUMN IN WINDOW 2
1 EPLINDX1 DS    CL1                 INDEX OFF/ON/FULL (0/1/2) WINDOW 1
1 EPLINDX2 DS    CL1                 INDEX OFF/ON/FULL (0/1/2) WINDOW 2
1 EPLPARLL EQU   *-EDTPARL
```

Specification of the IEDTPARL macro with VERSION=3 generates the control block EDTPARL in the following form:

```
           IEDTPARL D,VERSION=3

1 EDTPARL  MFPRE DNAME=EDT, MF=D, PREFIX=*
2 EDTPARL  DSECT,
2                 *,##### PREFIX=I, MACID= #####
1 *NAME     IDLKG ID=EDT, SECT=&D, VER=166

1 *-------------------------EDT UNIT NUMBER, EDTPARL VERSION NUMBER-------
1 EPLUNITC EQU   66                  EDT UNIT NUMBER
1 EPLVERSC EQU   3                   EDTPARL VERSION NUMBER
1 *                        *---- CONTROL BLOCK HEADER -------------------
1 EPLFHE   DS    0XL8                GENERAL OPERAND LIST HEADER
1 EPLIFID  DS    0A                  INTERFACE IDENTIFIER
1 EPLUNIT  DC    AL2(EPLUNITC)       UNIT NUMBER
1          DS    AL1                 RESERVED
1 EPLVERS  DC    AL1(EPLVERSC)       FUNCTION INTERFACE VERSION NUMBER
1          DS    A                   RESERVED
1 *                        *---- OUTPUT FIELDS -------------------------
1 EPLVPOS  DS    CL8                 FIRST LINE IN WINDOW
1 EPLHPOS  DS    H                   FIRST COLUMN IN WINDOW
1 EPLRLIM  DS    H                   MAX RECORD-LENGTH IN F-MODE
1 EPLINF   DS    CL1                 INF ON/OFF  (1/0)
1 EPLLOW   DS    CL1                 LOWER ON/OFF (1/0)
1 EPLHEX   DS    CL1                 HEX ON/OFF (1/0)
1 EPLEDL   DS    CL1                 EDIT LONG ON/OFF (1/0)
1 EPLSCALE DS    CL1                 SCALE ON/OFF (1/0)
1 EPLPROT  DS    CL1                 PROTECTION ON/OFF (1/0)
1 EPLSTRUC DS    CL1                 STRUCTURE SYMBOL
1 EPLOPEN  DS    CL1                 OPEN FLAG:   (I/P/R/S/X/0)
1 EPLEMPTY DS    CL1                 EMPTY FLAG
1 EPLMODIF DS    CL1                 MODIFIED FLAG
1 EPLSTDF  DS    CL54                STANDARD FILENAME
1 EPLSTDL  DS    CL54                STANDARD LIBRARY NAME
1 EPLSTDT  DS    CL8                 STANDARD PLAM TYPE
1 EPLSTCOD DS    CL1                 STANDARD CODE (E/I)
1          DS    CL3                 RESERVED
1 EPLVPOS1 DS    CL8                 FIRST LINE IN WINDOW 1
1 EPLHPOS1 DS    H                   FIRST COLUMN IN WINDOW 1
1 EPLVPOS2 DS    CL8                 FIRST LINE IN WINDOW 2
1 EPLHPOS2 DS    H                   FIRST COLUMN IN WINDOW 2
1 EPLINDX1 DS    CL1                 INDEX OFF/ON/FULL (0/1/2) WINDOW 1
1 EPLINDX2 DS    CL1                 INDEX OFF/ON/FULL (0/1/2) WINDOW 2
1 EPLOPENC DS    XL260               COMMON AREA FOR FILE DESCRIPTION
1 EPLOPEND EQU   *                   END OF COMMON AREA
1          ORG   EPLOPENC            DESCRIPTION OF OPENED DATA FILE
```

```
1 EPLOPNFL DS    CL54                   NAME OF OPENED FILE/PLAM LIBRARY
1 EPLOPNE  DS    CL64                   NAME OF OPENED PLAM ELEMENT
1 EPLOPNV  DS    CL24                   VERSION OF OPENED PLAM ELEMENT
1 EPLOPNT  DS    CL8                    TYP OF OPENED PLAM ELEMENT
           ORG   EPLOPENC               DESCRIPTION OF OPENED UFS FILE
1 EPLOPNX  DS    CL256                  NAME OF OPENED UFS FILE
1 EPLOPNXC DS    CL1                    CODE OF OPENED UFS FILE (E/I)

           ORG   EPLOPEND
1 *
1          DS    CL8                    RESERVED
1 EPLPARLL EQU   *-EDTPARL
```

Depending on the Open flag EPLOPEN the field EPLOPENC contains the description of the file opened with @OPEN or @XOPEN or of the PLAM element.

Meanings of the control block fields		**Length (bytes)**	**Parameter type**	
			Call	**Return**
EPLFHE	Header	(8)		
EPLIFID	Header input fields	(4)*		
EPLUNIT	Identification of EDT (66)	2	C	
EPLVERS	Update status of the control block	1	C	
EPLVPOS	1st line in the window (00000001..99999999)	8		R
EPLHPOS	1st column in the window (1..256)	2		R
EPLRLIM	Maximum record length, F mode (1..256)	2		R
EPLINF	Information On/Off (1/0)	1		R
EPLLOW	Lower On/Off (1/0)	1		R
EPLHEX	Hex On/Off (1/0)	1		R
EPLEDL	Edit Long On/Off (1/0)	1		R
EPLSCALE	Scale On/Off (1/0)	1		R
EPLPROT	Protection On/Off (1/0)	1		R
EPLSTRUC	Structure symbol	1		R

EPLOPEN	@OPEN display:
	R = ISAM real
	P = PLAM
	I = ISAM virtual
	S = SAM virtual
	X = POSIX
	0 = NO @OPEN
		1		R

Meanings of the control block fields		**Length (bytes)**	**Parameter type**	
			Call	**Return**
EPLSTCOD	Code default = E: EBCDIC I: ISO	1		R
EPLEMPTY	File empty Yes/No (1/0)	1		R
EPLMODIF	File changed Yes/No (1/0)	1		R
EPLSTDF	Standard file name	54		R
EPLSTDL	Standard library name	54		R
EPLSTDT	Standard type	8		R
EPLOPNC	EPLOPEN = R/I/S: DMS file name P: Description of the PLAM element X: Description of POSIX file	260		R
EPLOPNFL	EPLOPEN = R: ISAM file name real opened P: PLAM library name I: ISAM file name S: SAM file name	54		R
EPLOPNE	EPLOPEN = P: PLAM element name	64		R
EPLOPNV	EPLOPEN = P: PLAM version number	24		R
EPLOPNT	EPLOPEN = P: PLAM type	8		R
EPLOPNX	EPLOPEN = X: POSIX file name	256		R

EPLOPNXC	EPLOPEN = X: Code feature of POSIX file
	E = EBCDIC code
	I = ISO code
		1		R

EPLVPOS1	First line number in window 1	8		R
EPLHPOS1	First column in window 1	2		R
EPLVPOS2	First line number in window 2	8		R
EPLHPOS2	First column in window 2	2		R
EPLINDX1	Index Off/On/Full, window 1 (0/1/2)	1		R
EPLINDX2	Index Off/On/Full, window 2 (0/1/2)	1		R

3.5 Include files for programming in C

To make it easier to call EDT from a C program, macros for defining, initializing and modifying the EDT control blocks are supplied as include files in the user macro library SYSLIB.EDT.166. The return codes are defined as symbolic constants.

The meaning and use of the control blocks is described above.

3.5.1 iedglcb.h

Definitions and macros for the EDTGLCB global control block and definition of symbolic constants for the return codes:

```
#ifndef _IEDGLCB_H
#define _IEDGLCB_H

#if 0
/*********************************************************************
 BEGIN-INTERFACE   IEDGLCB

 TITLE             (/ EDT Global Control Block /)
 NAME              IEDGLCB.H
 DOMAIN            EDT
 LANGUAGE          C
 COPYRIGHT         (C) Siemens Nixdorf Informationssysteme AG 1995
                       ALL RIGHTS RESERVED
 COMPILATION-SCOPE USER
 INTERFACE-TYPE    LAYOUT
 RUN-CONTEXT       TU
 PURPOSE           (/ Definition of global control block.
                                                           /)
 REMARKS           (/
                    /)

---------------------------------------------------------------------


 VERSION           001
 CRDATE            1995-07-25
 AUTHOR            (/ Mondok D. PSE DKM313/)
 UPDATE            (/ Initial definition  /)

 END-INTERFACE     IEDGLCB.
*********************************************************************/
#endif

/* special values in MAINCODE                                          */
/* EDT call                                                    */
#define IEDGLCBcmd_no_error          0  /* successful processing      */
#define IEDGLCBcmd_syntax_error      8  /* syntax error in command    */
#define IEDGLCBcmd_runtime_error    12  /* runtime error in command   */
#define IEDGLCBcmd_unrec_edt_error  16  /* unrecoverable EDT error    */
#define IEDGLCBcmd_unrec_sys_error  20  /* unrecoverable system error */
#define IEDGLCBcmd_unrec_user_error 24  /* unrecoverable user error   */
#define IEDGLCBcmd_parameter_error  32  /* parameter error            */
#define IEDGLCBcmd_reqm_error       36  /* not enough space available */
#define IEDGLCBcmd_version_error    40  /* version error              */
#define IEDGLCBcmd_abnormal_error   44  /* abnormal halt by user      */

/* EDT access method                                           */
#define IEDGLCBacc_no_error          0  /* successful processing      */
#define IEDGLCBacc_access_error      4  /* access error               */
#define IEDGLCBacc_unrec_edt_error  16  /* unrecoverable EDT error    */
#define IEDGLCBacc_unrec_sys_error  20  /* unrecoverable system error */
#define IEDGLCBacc_unrec_user_error 24  /* unrecoverable user error   */
#define IEDGLCBacc_parameter_error  32  /* parameter error            */
#define IEDGLCBacc_reqm_error       36  /* not enough space available */

/* error classes in SUBCODE1                                   */
/* MAINCODE:   IEDGLCBcmd_no_error                             */
#define IEDGLCBno_error              0  /* successful processing      */
#define IEDGLCBhalt                  4  /* halt entered               */
#define IEDGLCBhalt_text             8  /* halt with text entered     */
#define IEDGLCBreturn               12  /* return entered             */
#define IEDGLCBreturn_text          16  /* return with text entered   */
#define IEDGLCBk1_key               20  /* return with k1             */
#define IEDGLCBignore_command       24  /* only in stmnt filter:      */
                                        /* statement to be ignore     */


/* MAINCODE:   IEDGLCBcmd_parameter_error                      */
#define IEDGLCBglcb_error            4  /* error in EDTGLCB           */
#define IEDGLCBupcb_error            8  /* error in EDTUPCB           */
#define IEDGLCBparameter_error      12  /* error in command parameter */
#define IEDGLCBmessage_error        16  /* error in message parameter */

/* MAINCODE:   IEDGLCBcmd_version_error                        */
#define IEDGLCBstandard_version      0  /* standard version returned  */
#define IEDGLCBno_version_returned   4  /* no version returned        */

/* MAINCODE:   IEDGLCBacc_no_error                             */
#define IEDGLCBacc_ok                0  /* no errror                  */
#define IEDGLCBnext_record           4  /* next record returned       */
#define IEDGLCBfirst_record          8  /* first record returned      */
#define IEDGLCBlast_record          12  /* last record returned       */
```

```
#define IEDGLCBfile_cleared         16  /* file cleard                 */
#define IEDGLCBcopy_buffer_cleared  20  /* copy buffer cleard          */

/* MAINCODE:   IEDGLCBacc_access_error                                 */
#define IEDGLCBput_record_truncated  4  /* put record truncated        */
#define IEDGLCBkey_truncated         8  /* key truncated (move mode)   */
#define IEDGLCBrecord_truncated     12  /* rec. truncated (move mode)  */
#define IEDGLCBfile_empty           16  /* file is empty               */
#define IEDGLCBno_marks             20  /* no marks in file            */
#define IEDGLCBfile_not_opened      24  /* file not opened             */
#define IEDGLCBfile_real_opened     28  /* file real opened (no marks)*/
#define IEDGLCBmark_not_found       32  /* mark not found   (IEDTPTM)  */
#define IEDGLCBkey_error            36  /* key error        (IEDTREN)  */
#define IEDGLCBmax_line_number      40  /* maximum line number reached*/
#define IEDGLCBrenumber_inhibited   44  /* renumber is inhibited       */
#define IEDGLCBfile_active          48  /* file is active              */

/* MAINCODE:   IEDGLCBacc_parameter_error                              */
#define IEDGLCBacc_glcb_error        4  /* error in EDTGLCB            */
#define IEDGLCBacc_amcb_error        8  /* error in EDTAMCB            */
#define IEDGLCBfilename_error       12  /* filename error              */
#define IEDGLCBacc_function_error   16  /* access function error       */
#define IEDGLCBkey_format_error     20  /* error in key format         */
#define IEDGLCBkey_length_error     24  /* error on key length         */
#define IEDGLCBrecord_length_error  28  /* error on record length      */
#define IEDGLCBmode_byte_error      32  /* error in transfer mode      */
#define IEDGLCBunit_version_error   36  /* error in version or unit    */



/* special values in KEY-CODE                                          */
#define IEDGLCBkey_code_DUE        102  /* DUE                         */
#define IEDGLCBkey_code_F1          91  /* F1                          */
#define IEDGLCBkey_code_F2          92  /* F2                          */
#define IEDGLCBkey_code_F3          93  /* F3                          */
#define IEDGLCBkey_code_K1          83  /* K1                          */


/* IEDGLCB parameter block                                             */

struct IEDGLCB_mdl {
/* interface identifier structure                                      */
        #pragma aligned 4
        unsigned short unit;            /* function unit number  : 66 */
        unsigned char function;         /* function number       :  0 */
        unsigned char version;          /* interface version no. :  1 */

/* returncode structure                                                */
```

```
        union /* rc */ {
                struct {
                        struct {
                                unsigned char subcode2;
                                unsigned char subcode1;
                        } subcode;
                        union /* mc */ {
                                unsigned short maincode;
                                struct {
                                unsigned char maincode2;
                                unsigned char maincode1;
                                } main_returncode;
                        } mc;
                } structured_rc;
                unsigned long rc_nbr;   /* general return code        */
        } rc;

/* info size or displacement of invalid command                       */
        union          {
                unsigned long memo_size;      /* information of       */
                                              /* memory size          */
                unsigned long displ_to_cmd;   /* displacement of      */
                                              /* invalid command      */
        } size_or_displacement;

/* return message field                                               */
        union          {
                struct {
                        unsigned short    rmsgl; /* message length    */
                        unsigned char rmsgf[80]; /* message field     */
                } structured_msg;
                unsigned char rmsg[82];       /* return message       */
        } return_message;

/* code of sending key                                                */
        unsigned char key_code;

/* indicator byte                                                     */
        struct {
           unsigned char not_used_1      :1; /* not used              */
           unsigned char not_used_2      :1; /* not used              */
           unsigned char reorg_allowed   :1; /* reorganisation allowed*/
           unsigned char not_used_3      :1; /* not used              */
           unsigned char stxit_allowed   :1; /* EDT STXIT allowed     */
           unsigned char data_initiated  :1; /* EDT data initiated    */
           unsigned char data_add_valid  :1; /* EDT data addr. valid  */
           unsigned char entry_add_valid :1; /* EDT entry addr. valid */
                 } indicator;
```

```
/* EDT entry address                                                   */
        void* EDT_entry;

/* EDT data address                                                    */
        void* EDT_data;

/* name of actual workfile                                             */
        unsigned char filename [8];

/* user parameter 1                                                    */
        union          {
        unsigned char user_param1_char[4];
        void*         user_param1_pointer;
        } user_param1;

/* user parameter 2                                                    */
        union          {
        unsigned char user_param2_char[4];
        void*         user_param2_pointer;
        } user_param2;

/* user parameter 3                                                    */
        union          {
        unsigned char user_param3_char[4];
        void*         user_param3_pointer;
        } user_param3;
};



/* macros for initialization, access, and modification                */

#define IEDGLCB_RC_NIL              -1
#define IEDGLCB_RC_NULL             0
#define IEDGLCB_UNIT_66             66
#define IEDGLCB_FUNCT_0             0
#define IEDGLCB_VERS_1              1
#define IEDGLCB_INIT                { IEDGLCB_UNIT_66, \
                                      IEDGLCB_FUNCT_0, \
                                      IEDGLCB_VERS_1, \
                                      IEDGLCB_RC_NULL }
#define IEDGLCB_UNIT                unit
#define IEDGLCB_FUNCT               function
#define IEDGLCB_VERS                version
#define IEDGLCB_RC_SUBCODE2         rc.structured_rc.subcode.subcode2
#define IEDGLCB_RC_SUBCODE1         rc.structured_rc.subcode.subcode1
#define IEDGLCB_RC_MAINCODE         rc.structured_rc.mc.maincode
```

```
#define IEDGLCB_RC_MAINCODE2       rc.structured_rc.mc.\
                                main_returncode.maincode2
#define IEDGLCB_RC_MAINCODE1       returncode.rc.structured_rc.mc.\
                                main_returncode.maincode1
#define IEDGLCB_RC_NBR             rc.rc_nbr
#define IEDGLCB_MOD_VERS(p,v)      p.IEDGLCB_VERS  = v
#define IEDGLCB_MOD_IFID(p,u,f,v) \
                                p.IEDGLCB_UNIT  = u , \
                                p.IEDGLCB_FUNCT = f , \
                                p.IEDGLCB_VERS  = v
#define IEDGLCB_MOD_RC(p,sc2,sc1,mrc) \
                                p.IEDGLCB_RC_SUBCODE2 = sc2 , \
                                p.IEDGLCB_RC_SUBCODE1 = sc1 , \
                                p.IEDGLCB_RC_MAINCODE = mrc
#define IEDGLCB_SET_RC_NIL(p)      p.IEDGLCB_RC_NBR = IEDGLCB_RC_NIL
#define IEDGLCB_SET_RC_NULL(p)     p.IEDGLCB_RC_NBR = IEDGLCB_RC_NULL

#endif          /* _IEDGLCB_H */
```

3.5.2 iedupcb.h

Definitions and macros for the EDTUPCB subroutine control block:

```
#ifndef _IEDUPCB_H
#define _IEDUPCB_H

#if 0
/**********************************************************************
 BEGIN-INTERFACE   IEDUPCB

 TITLE             (/ EDT subprogram control block /)
 NAME              IEDUPCB.H
 DOMAIN            EDT
 LANGUAGE          C
 COPYRIGHT         (C) Siemens Nixdorf Informationssysteme AG 1995
                       ALL RIGHTS RESERVED
 COMPILATION-SCOPE USER
 INTERFACE-TYPE    LAYOUT
 RUN-CONTEXT       TU
 PURPOSE           (/ Definition of subprogram control block.
                                                         /)
 REMARKS           (/
                    /)
----------------------------------------------------------------------

 VERSION           001
 CRDATE            1995-07-25
 AUTHOR            (/ Mondok D. PSE DKM313/)
 UPDATE            (/ Initial definition  /)

 END-INTERFACE     IEDUPCB.
**********************************************************************/
#endif


/* IEDUPCB parameter block                                            */

struct IEDUPCB_mdl {
/* interface identifier structure                                     */
        #pragma aligned 4
        unsigned short unit;          /* function unit number  : 66 */
        unsigned char function;       /* function number       :  0 */
        unsigned char version;        /* interface version no. :  1 */

/* returncode unused                                                  */
/* returncode will be returned in control block IEDGLCB               */
        unsigned long rc_nbr;
/* inhibit flag byte                                                  */
        union  {
           struct {                            /* INHIBIT :           */
              unsigned char not_used_2      :1; /* not used            */
              unsigned char no_text_at_exit :1;
                                     /* @HALT <text> / @RET <text>     */
              unsigned char no_edit_only    :1; /* @EDIT ONLY          */
              unsigned char no_edit         :1; /* @EDIT               */
              unsigned char no_user_prog    :1; /* @RUN                */
              unsigned char no_bkpt         :1; /* @SYSTEM             */
              unsigned char no_cmd          :1; /* @SYSTEM <string>    */
              unsigned char no_exec         :1; /* @EXEC/@LOAD         */
             } bit;
           unsigned char byte;
        } inhibit;

/* reserve                                                            */
        unsigned char reserve  [3];

};

/* macros for initialization, access, and modification                */

#define IEDUPCB_RC_NULL            0
#define IEDUPCB_UNIT_66            66
#define IEDUPCB_FUNCT_0            0
#define IEDUPCB_VERS_STD           2
#define IEDUPCB_NO_INHIBIT         0
#define IEDUPCB_INIT               { IEDUPCB_UNIT_66, \
                                     IEDUPCB_FUNCT_0, \
                                     IEDUPCB_VERS_STD, \
                                     IEDUPCB_RC_NULL }
#define IEDUPCB_UNIT               unit
#define IEDUPCB_FUNCT              function
#define IEDUPCB_VERS               version
#define IEDUPCB_RC_NBR             rc_nbr
#define IEDUPCB_MOD_VERS(p,v)      p.IEDUPCB_VERS  = v
#define IEDUPCB_MOD_IFID(p,u,f,v) \
                                p.IEDUPCB_UNIT  = u , \
                                p.IEDUPCB_FUNCT = f , \
                                p.IEDUPCB_VERS  = v
#define IEDUPCB_SET_NO_INHIBIT(p)  p.inhibit.byte = IEDUPCB_NO_INHIBIT
#define IEDUPCB_SET_RC_NULL(p)     p.IEDUPCB_RC_NBR = IEDUPCB_RC_NULL

#endif          /* _IEDUPCB_H */
```

3.5.3 iedambc.h

Definitions and macros for the EDTAMCB record access control block:

```
#ifndef _IEDAMCB_H
#define _IEDAMCB_H

#if 0
/**********************************************************************
 BEGIN-INTERFACE   IEDAMCB

 TITLE             (/ EDT Access Method Control Block /)
 NAME              IEDAMCB.H
 DOMAIN            EDT
 LANGUAGE          C
 COPYRIGHT         (C) Siemens Nixdorf Informationssysteme AG 1995
                       ALL RIGHTS RESERVED
 COMPILATION-SCOPE USER
 INTERFACE-TYPE    LAYOUT
 RUN-CONTEXT       TU
 PURPOSE           (/ Definition of access method control block.
                                                             /)
 REMARKS           (/
                    /)


----------------------------------------------------------------------


 VERSION           001
 CRDATE            1995-07-25
 AUTHOR            (/ Mondok D. PSE DKM313/)
 UPDATE            (/ Initial definition  /)

 END-INTERFACE     IEDAMCB.
**********************************************************************/
#endif


/* IEDAMCB parameter block                                             */

struct IEDAMCB_mdl {
/* interface identifier structure                                      */
        #pragma aligned 4
        unsigned short unit;           /* function unit number  : 66 */
        unsigned char function;        /* function number       :  0 */
        unsigned char version;         /* interface version no. :  1 */
```

```
/* returncode structure                                                 */
        union /* rc */ {
                struct {
                        struct {
                                unsigned char subcode2;
                                unsigned char subcode1;
                        } subcode;
                        union /* mc */ {
                                unsigned short maincode;
                                struct {
                                unsigned char maincode2;
                                unsigned char maincode1;
                                } main_returncode;
                        } mc;
                } structured_rc;
                unsigned long rc_nbr;   /* general return code          */
        } rc;

/* transfer mode flag byte                                              */
        union       {
             struct {
                     unsigned char not_used_1 :5 ;/* not used           */
                     unsigned char locate     :1 ;/* locate mode        */
                     unsigned char not_used_2 :2 ;/* not used           */
             } mode_bits;
             unsigned char mode_byte;        /* mode byte               */
        } mode_flag;

/* flag byte                                                            */
        union       {
             struct {
                 unsigned char not_used       :6 ;/* not  used          */
                 unsigned char inh_set_modify :1 ;/* inhibit setting    */
                                                 /*  modify flag        */
                 unsigned char ign_mark13     :1 ;/* ignore mark 13     */
                } flag_bits;
             unsigned char flag_byte;
        } flag;

/* input parameters                                                     */
        unsigned char filename [8];          /* workfile                */
        unsigned long displacement;          /* displacement            */
        unsigned short length_key1;          /* length of key1          */
        unsigned short length_key2;          /* length of key2          */

/* input parameters (only in move mode)                                 */
        unsigned short length_key_outbuffer; /* length of key output    */
                                             /* buffer                  */
        unsigned short length_rec_outbuffer; /* length of rec output  */
                                             /* buffer                */

/* input/output parameters                                            */
        unsigned short length_key;           /* length of key         */
        unsigned short length_rec;           /* length of record      */

/* marks                                                              */
        union  {
                unsigned short mark_field;
                struct {
                   union  {
                        unsigned char mark2;  /* upper marks          */
                        struct {
                                unsigned char mark_15 :1 ;/* mark 15  */
                                unsigned char mark_14 :1 ;/* mark 14  */
                                unsigned char mark_13 :1 ;/* mark 13  */
                                unsigned char mark_12 :1 ;/* mark 12  */
                                unsigned char mark_11 :1 ;/* mark 11  */
                                unsigned char mark_10 :1 ;/* mark 10  */
                                unsigned char mark_9  :1 ;/* mark 9   */
                                unsigned char mark_8  :1 ;/* mark 8   */
                        } mark2_bits;
                   } upper_marks;
                   union  {
                        unsigned char mark1;  /* lower marks          */
                        struct {
                                unsigned char mark_7  :1 ;/* mark 7   */
                                unsigned char mark_6  :1 ;/* mark 6   */
                                unsigned char mark_5  :1 ;/* mark 5   */
                                unsigned char mark_4  :1 ;/* mark 4   */
                                unsigned char mark_3  :1 ;/* mark 3   */
                                unsigned char mark_2  :1 ;/* mark 2   */
                                unsigned char mark_1  :1 ;/* mark 1   */
                                unsigned char mark_0  :1 ;/* mark 0   */
                        } mark1_bits;
                   } lower_marks;
                } mark_bytes;
        } marks;

/* reserve                                                            */
        unsigned char reserve  [2];

};
```

```
/* macros for initialization, access, and modification               */

#define IEDAMCB_RC_NIL              -1
#define IEDAMCB_RC_NULL             0
#define IEDAMCB_UNIT_66             66
#define IEDAMCB_FUNCT_0             0
#define IEDAMCB_VERS_STD            1
#define IEDAMCB_NO_MARKS            0
#define IEDAMCB_INIT                { IEDAMCB_UNIT_66, \
                                      IEDAMCB_FUNCT_0, \
                                      IEDAMCB_VERS_STD, \
                                      IEDAMCB_RC_NULL }
#define IEDAMCB_UNIT                unit
#define IEDAMCB_FUNCT               function
#define IEDAMCB_VERS                version
#define IEDAMCB_RC_SUBCODE2         rc.structured_rc.subcode.subcode2
#define IEDAMCB_RC_SUBCODE1         rc.structured_rc.subcode.subcode1
#define IEDAMCB_RC_MAINCODE         rc.structured_rc.mc.maincode
#define IEDAMCB_RC_MAINCODE2        rc.structured_rc.mc.\
                                main_returncode.maincode2
#define IEDAMCB_RC_MAINCODE1        returncode.rc.structured_rc.mc.\
                                main_returncode.maincode1
#define IEDAMCB_RC_NBR              rc.rc_nbr
#define IEDAMCB_MARKS               marks.mark_field
#define IEDAMCB_MOD_VERS(p,v)       p.IEDAMCB_VERS  = v
#define IEDAMCB_SET_NO_MARKS(p)     p.IEDAMCB_MARKS = IEDAMCB_NO_MARKS
#define IEDAMCB_MOD_IFID(p,u,f,v) \
                                p.IEDAMCB_UNIT  = u , \
                                p.IEDAMCB_FUNCT = f , \
                                p.IEDAMCB_VERS  = v
#define IEDAMCB_MOD_RC(p,sc2,sc1,mrc) \
                                p.IEDAMCB_RC_SUBCODE2 = sc2 , \
                                p.IEDAMCB_RC_SUBCODE1 = sc1 , \
                                p.IEDAMCB_RC_MAINCODE = mrc
#define IEDAMCB_SET_RC_NIL(p)       p.IEDAMCB_RC_NBR = IEDAMCB_RC_NIL
#define IEDAMCB_SET_RC_NULL(p)      p.IEDAMCB_RC_NBR = IEDAMCB_RC_NULL

#endif          /* _IEDAMCB_H */
```

3.5.4 iedparg.h

Definitions and macros for the EDTPARG control block (information on the global status):

```
#ifndef _IEDPARG_H
#define _IEDPARG_H

#if 0
/**********************************************************************
 BEGIN-INTERFACE   IEDPARG

 TITLE             (/ EDT control block for global status information /)
 NAME              IEDPARG.H
 DOMAIN            EDT
 LANGUAGE          C
 COPYRIGHT         (C) Siemens Nixdorf Informationssysteme AG 1995
                       ALL RIGHTS RESERVED
 COMPILATION-SCOPE USER
 INTERFACE-TYPE    LAYOUT
 RUN-CONTEXT       TU
 PURPOSE           (/ Definition of control block IEDTPARG.
                                                             /)
 REMARKS           (/
                    /)

----------------------------------------------------------------------

 VERSION           001
 CRDATE            1995-07-25
 AUTHOR            (/ Mondok D. PSE DKM313/)
 UPDATE            (/ Initial definition  /)

 END-INTERFACE     IEDPARG.
**********************************************************************/
#endif


/* special values in EDT_mode                                          */
#define IEDPARGmode_fullscreen     'F'  /* full screen mode            */
#define IEDPARGmode_line           'L'  /* line mode                   */
#define IEDPARGmode_control        'C'  /* user control                */


/* IEDPARG parameter block                                             */
```

```
struct IEDPARG_mdl {
/* interface identifier structure                                        */
        #pragma aligned 4
        unsigned short unit;            /* function unit number  : 66 */
        unsigned char function;         /* function number       :  0 */
        unsigned char version;          /* interface version no. :  1 */

/* returncode unused                                                     */
/* returncode will be returned in control block IEDGLCB                  */
        unsigned long rc_nbr;

/* output fields                                                         */
        unsigned char EDT_mode;              /* edt mode              */
        unsigned char command_symbol;        /* actual'@'             */
        unsigned short size_window1;         /* size of window1       */
        unsigned short size_window2;         /* size of window2       */
        unsigned char file_in_window1 [8];   /* work file in window1  */
        unsigned char file_in_window2 [8];   /* work file in window2  */
        unsigned char ccs_name        [8];   /* coded character set   */


};

/* macros for initialization, access, and modification                   */

#define IEDPARG_RC_NULL           0
#define IEDPARG_UNIT_66           66
#define IEDPARG_FUNCT_0           0
#define IEDPARG_VERS_STD          1
#define IEDPARG_INIT              { IEDPARG_UNIT_66, \
                                    IEDPARG_FUNCT_0, \
                                    IEDPARG_VERS_STD, \
                                    IEDPARG_RC_NULL }
#define IEDPARG_UNIT              unit
#define IEDPARG_FUNCT             function
#define IEDPARG_VERS              version
#define IEDPARG_RC_NBR            rc.rc_nbr
#define IEDPARG_MOD_VERS(p,v)     p.IEDPARG_VERS  = v
#define IEDPARG_MOD_IFID(p,u,f,v) \
                               p.IEDPARG_UNIT  = u , \
                               p.IEDPARG_FUNCT = f , \
                               p.IEDPARG_VERS  = v
#define IEDPARG_SET_RC_NULL(p)    p.IEDPARG_RC_NBR = IEDPARG_RC_NULL

#endif          /* _IEDPARG_H */
```

3.5.5 iedparl.h

Definitions and macros for the EDTPARL control block (information on the local file status):

```
#ifndef _IEDPARL_H
#define _IEDPARL_H

#if 0
/**********************************************************************
 BEGIN-INTERFACE   IEDPARL

 TITLE             (/ EDT control block for local status information /)
 NAME              IEDPARL.H
 DOMAIN            EDT
 LANGUAGE          C

 COPYRIGHT         (C) Siemens Nixdorf Informationssysteme AG 1995
                       ALL RIGHTS RESERVED
 COMPILATION-SCOPE USER
 INTERFACE-TYPE    LAYOUT
 RUN-CONTEXT       TU
 PURPOSE           (/ Definition of control block IEDTPARL.
                                                              /)
 REMARKS           (/
                    /)

----------------------------------------------------------------------
 VERSION           001
 CRDATE            1995-07-25
 AUTHOR            (/ Mondok D. PSE DKM313/)
 UPDATE            (/ Initial definition  /)

 END-INTERFACE     IEDPARL.
**********************************************************************/
#endif


/* special values in open_flag                                      */
#define IEDPARLopen_isam          'I'  /* ISAM file virtually opened */
#define IEDPARLopen_plam          'P'  /* PLAM element opened        */
#define IEDPARLopen_real          'R'  /* ISAM file really opened    */
#define IEDPARLopen_sam           'S'  /* SAM file virtually opened  */
#define IEDPARLopen_ufs           'X'  /* UFS file virtually opened  */
#define IEDPARLopen_no            'O'  /* no file opened             */
```

```
/* special values in index                                           */
#define IEDPARLindex_off            '0'  /* INDEX OFF                 */
#define IEDPARLindex_on             '1'  /* INDEX ON                  */
#define IEDPARLindex_full           '2'  /* EDIT FULL                 */

/* IEDPARL parameter block                                           */

struct IEDPARL_mdl {
/* interface identifier structure                                    */
        #pragma aligned 4
        unsigned short unit;             /* function unit number  : 66 */
        unsigned char function;          /* function number       :  0 */
        unsigned char version;           /* interface version no. :  3 */

/* returncode unused                                                 */
/* returncode will be returned in control block IEDGLCB              */
        unsigned long rc_nbr;

/* output fields                                                     */
        unsigned char first_line_window [8]; /* number of first line  */
                                            /* in window              */
        unsigned short first_col_window;    /* first column in window  */
        unsigned short record_length_max;   /* max. record length in   */
                                            /* in full screen mode     */
        unsigned char par_inf;              /* INF         on/off (1/0)*/
        unsigned char par_low;              /* LOWER       on/off (1/0)*/
        unsigned char par_hex;              /* HEX         on/off (1/0)*/
        unsigned char par_edit_long;        /* EDIT-LONG   on/off (1/0)*/
        unsigned char par_scale;            /* SCALE       on/off (1/0)*/
        unsigned char par_protection;       /* PROTECTION  on/off (1/0)*/
        unsigned char structure_symbol;     /* structure symbol        */
        unsigned char open_flag;            /* open flag (I/P/R/S/X/O) */
        unsigned char empty_flag;           /* empty          y/n (1/0)*/
        unsigned char modified_flag;        /* modified       y/n (1/0)*/
        unsigned char std_file     [54]; /* standard file name         */
        unsigned char std_library  [54]; /* standard library name      */
        unsigned char std_plam_type [8]; /* standard plam type         */
        unsigned char std_code;          /* standard code      (E/I)  */
        unsigned char not_used1     [3]; /* reserved                   */
        unsigned char first_line1   [8]; /* number of first line       */
                                         /* in window1                 */
        unsigned short first_col1;       /* first column in window1    */
        unsigned char first_line2   [8]; /* number of first line       */
                                         /* in window2                 */
        unsigned short first_col2;       /* first column in window2    */
        unsigned char index_window1;     /* INDEX OFF/ON/FULL (0/1/2) */
        unsigned char index_window2;     /* INDEX OFF/ON/FULL (0/1/2) */
```

```
   union {             /* description of opened data file        */
              unsigned char common_area [260];    /* common area     */
              struct {
                  unsigned char file_name [54]; /* name of opened    */
                                                /* file or plam lib. */
                  unsigned char plam_elem [64]; /* name of plam elem.*/
                  unsigned char plam_vers [24]; /* name of plam vers.*/
                  unsigned char plam_type  [8]; /* plam type         */
              } file_or_plam_elem;
              struct {
                  unsigned char ufs_name [256]; /* name of opened    */
                                                /* ufs file          */
                  unsigned char code;           /* code of opened    */
                                                /* ufs file          */
              } ufs_file;
       } file_description;

       unsigned char not_used2     [8]; /* reserved                  */

};

/* macros for initialization, access, and modification              */

#define IEDPARL_RC_NULL            0
#define IEDPARL_UNIT_66            66
#define IEDPARL_FUNCT_0            0
#define IEDPARL_VERS_STD           3
#define IEDPARL_INIT               { IEDPARL_UNIT_66, \
                                     IEDPARL_FUNCT_0, \
                                     IEDPARL_VERS_STD, \
                                     IEDPARL_RC_NULL }
#define IEDPARL_UNIT               unit
#define IEDPARL_FUNCT              function
#define IEDPARL_VERS               version
#define IEDPARL_RC_NBR             rc.rc_nbr
#define IEDPARL_MOD_VERS(p,v)      p.IEDPARL_VERS  = v
#define IEDPARL_MOD_IFID(p,u,f,v) \
                                p.IEDPARL_UNIT  = u , \
                                p.IEDPARL_FUNCT = f , \
                                p.IEDPARL_VERS  = v
#define IEDPARL_SET_RC_NULL(p)     p.IEDPARL_RC_NBR = IEDPARL_RC_NULL

#endif          /* _IEDPARL_H */
```

3.6 Examples

3.6.1 EDT as a subroutine of a COBOL program

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. EDTUP.
*
*---------------------------------------------------------------*
*  EXAMPLE: CALLING EDT AS A SUBROUTINE FROM A COBOL PROGRAM
*
*  THE CALL IS EXECUTED USING THE CMD FUNCTION
*---------------------------------------------------------------*
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
 TERMINAL IS T.
 DATA DIVISION.
*================================================================
*                       DATA SECTION
*================================================================
 WORKING-STORAGE SECTION.
*---------------------------------------------------------------*
* DATA WHICH IS PASSED TO EDT:
*       1) GLOBAL CONTROL BLOCK (EDTGLCB)
*       2) SUBROUTINE CONTROL BLOCK (EDTUPCB)
*       3) COMMAND (SEQUENCE) FOR "IEDTCMD" FUNCTION
*       4) MESSAGE FOR WINDOW 1
*       5) MESSAGE FOR WINDOW 2
*---------------------------------------------------------------*
 01  EDTGLCB.
     02  EGLFHE.
         03  EGLUNIT             PIC XX.
         03  RESERVED            PIC X.
         03  EGLVERS             PIC X.
     02  EGLRETC.
         03  EGLSRET.
             04  EGLSR2          PIC X.
             04  EGLSR1          PIC X.
         03  EGLMRET             PIC XX.
     02  EGLINFM.
         03  EGLCMDS             PIC X(4).
         03  EGLRMSG.
             04  EGLRMSGL        PIC XX.
             04  EGLRMSGF        PIC X(80).
      02 RESERVED                PIC X(1).
      02 EGLINDB                PIC X.
      02 EGLENTRY               PIC X(4).
      02 EGLDATA                PIC X(4).
      02 EGLFILE                PIC X(8).
      02 EGLUSER                PIC X(12).
*---------------------------------------------------------------*
 01  EDTUPCB.
     02  EUPFHE.
         03  EUPIFID.
             04  EUPUNIT       PIC XX.
             04  RESERVED      PIC X.
             04  EUPVERS       PIC X.
             04  RESERVED      PIC X(4).
     02  EUPINHBT              PIC X.
     02  RESERVED              PIC X(3).
*---------------------------------------------------------------*
 01  STATEMENT-SEQ-TO-BE-EXECUTED.
     02  LENGTH1  PIC 99 COMP VALUE IS 52.
     02  FILLER  PIC XX.
     02  SEQUENCE PIC X(48) VALUE IS
         "CREATE 5.00 'TEXT';PAR SPLIT = 10 $1;DIALOG;HALT".
*---------------------------------------------------------------*
 01  MESSAGE-FOR-WORK-WINDOW-1.
     02  LENGTH2  PIC 99 COMP VALUE IS 49.
     02  FILLER  PIC XX.
     02  MESSAGE1 PIC X(45) VALUE IS
         "MESSAGE IN WINDOW 1: EDT HAS BEEN CALLED.....".
*---------------------------------------------------------------*
 01  MESSAGE-FOR-WORK-WINDOW-2.
     02  LENGTH3  PIC 99 COMP VALUE IS 49.
     02  FILLER  PIC XX.
     02  MESSAGE PIC X(45) VALUE IS
         "MESSAGE IN WINDOW 2: ASA SUBROUTINE BY COBOL.".
*---------------------------------------------------------------*
* VARIABLES FOR INITIALIZING THE
*      UNIT NUMBER (X'66') AND THE VERSION NUMBER (X'02')
* IN EDTGLCB AND EDTUPCB.
*---------------------------------------------------------------*
 01  BINX66   PIC 99 COMP VALUE IS 66.
 01  CHARX66  REDEFINES BINX66 PIC XX.
*
 01  BINX1    PIC 99 COMP VALUE IS 1.
 01  CHAR1    REDEFINES BINX1.
     02 FILLER PIC X.
     02 CHARX1 PIC X.
*
 01  BINX2    PIC 99 COMP VALUE IS 2.
 01  CHAR2    REDEFINES BINX2.
```

```
     02 FILLER PIC X.
     02 CHARX2 PIC X.
*================================================================
*                       STATEMENT SECTION
*================================================================
 PROCEDURE DIVISION.
*---------------------------------------------------------------*
 PARAMETERLIST-INITIALIZATION.
* PLACE UNIT AND VERSION NUMBERS IN CONTROL BLOCKS
      MOVE CHARX66 TO EGLUNIT.
      MOVE CHARX66 TO EUPUNIT.
      MOVE CHARX1  TO EGLVERS.
      MOVE CHARX2  TO EUPVERS.
* DISABLE ALL POSSIBLE STATEMENTS (@EDIT,@SYS,@EXEC,...)
      MOVE HIGH-VALUE TO EUPINHBT.
*---------------------------------------------------------------*
 EDT-CALL.
      CALL "IEDTCMD" USING EDTGLCB,
                           EDTUPCB,
                           STATEMENT-SEQ-TO-BE-EXECUTED,
                           MESSAGE-FOR-WORK-WINDOW-1,
                           MESSAGE-FOR-WORK-WINDOW-2.
      DISPLAY "EDT HAS BEEN CALLED BY A COBOL PROGRAM"
               UPON T.
*---------------------------------------------------------------*
 RETURN CODE HANDLING.
*---------------------------------------------------------------*
* THE INFORMATION RETURNED BY EDT IS EVALUATED HERE:
*       RETURN CODES (EGLSRET,EGLMRET)
*       EDT MESSAGE (EGLCMDS,EGLRMGF,..)
*---------------------------------------------------------------*
 PROGAM-END.
     STOP RUN.
```

3.6.2 EDT as a subroutine of an Assembler program

```
**********************************************************************
* TESTUP :  EXAMPLE OF A SUBROUTINE APPLICATION                      *
*                                                                    *
**********************************************************************
* FUNCTION: THIS PROGRAM USES EDT TO ENTER DATA. THE MAIN PROGRAM    *
*           CHECKS THE RECORDS WHICH HAVE BEEN INPUT FOR THE CORRECT *
*           RECORD LENGTH. IN THE EVENT OF ERRORED RECORDS           *
*           A CORRECTION DIALOG IS INITIATED (AFTER APPROPRIATE      *
*           EDITING).                                                *
*                                                                    *
* 1)          INITIALIZATION OF PARAMETERS                           *
* 2) IEDTINF: READ THE EDT VERSION NUMBER                            *
* 3) IEDTCMD: SET EDT DEFAULT VALUES                                 *
*             SWITCH TO THE (CORRECTION) DIALOG                      *
* 4) IEDTGET: QUERY GLOBAL FILE STATUS                               *
* 5) IEDTDEL: CLEAR LOGGING FILE $1                                  *
* 6) IEDTGTM: READ MARKED RECORD FROM $0                             *
*    IEDTPTM: DELETE RECORD MARK                                     *
* 7) IEDTGET: READ RECORD FROM $0 WITH LENGTH CHECK (MAX = 40)       *
*    IEDTPTM: MARK ERRORED RECORD WITH MARK 14                       *
*    IEDTPUT: MOVE ERRORED RECORD TO $1                              *
* 8) IEDTREN: RENUMBER LINES IN $1                                   *
* 9)          IN THE CASE OF "HALT" AND AN ERROR-FREE FILE ----> END *
*             OTHERWISE INTERACTIVE CORRECTION (3)                   *
* 10)         EDT CALLS + ERROR RECOVERY                             *
*                                                                    *
**********************************************************************
TESTUP   START
TESTUP   AMODE ANY
TESTUP   RMODE ANY
         GPARMOD 31
         PRINT NOGEN
         REGS  ,
         BALR  R10,0
         USING *,R10
         SPACE ,
* 1) INITIALIZATION OF PARAMETERS ************************************
         SPACE ,
         LA    R13,SAVEAREA
*--- EDTGLCB --------------------------------------------------------*
         MVI   EGLINDB,X'00'           CLEAR EDT INDICATORS
*--- EDTUPCB --------------------------------------------------------*
         OI    EUPINHBT,EUPN@EDT       NO L MODE DIALOG
         OI    EUPINHBT,EUPN@EDO       NO L MODE PROCEDURE
         OI    EUPINHBT,EUPNTXT        NO <TEXT> ON TERMINATION
```

```
*--- EDTAMCB ---------------------------------------------------------*
         LH    R9,=H'8'
         STH   R9,EAMLKEY1             INDEX LENGTHS
         STH   R9,EAMLKEY2
         STH   R9,EAMLKEY
         STH   R9,EAMPKEY              INDEX BUFFER
         LH    R9,=H'256'
         STH   R9,EAMPREC              RECORD BUFFER
         SPACE ,
* 2) READ THE EDT VERSION NUMBER *************************************
         SPACE ,
         BAL   R11,INFCALL
         MVC   VERSION(EGLVERSL),EGLRMSGF
         SPACE ,
* 3) (CORRECTION) DIALOG *********************************************
         SPACE ,
DIALOG   BAL   R11,CMDCALL
         MVC   AKTHALT,EGLSR1          SAVE TERMINATION
         XC    FLAG,FLAG               ERROR INDICATOR
         SPACE ,
* 4) QUERY GLOBAL FILE STATUS ****************************************
         SPACE ,
         MVI   EAMMODB,EAMMOVM         MOVE MODE
         MVC   EAMFILE,=CL8'G'         REQUEST GLOBAL VALUES
         XC    EAMDISP,EAMDISP
         MVC   EAMPREC(2),=AL2(EPGPARGL) SUPPLY LENGTH VALUE
         BAL   R11,GETGCALL
         CLC   EPGFILE1,=CL8'$0'       CHECK WINDOW 1
         BE    WINDOW2
         CLC   EPGFILE1,=CL8'$1'
         BNE   INFO                    --> MESSAGE
WINDOW2  LH    R9,EPGWDS2              DOES WINDOW 2 EXIST?
         LTR   R9,R9
         BE    DEL$1                   NO
         CLC   EPGFILE2,=CL8'$1'       CHECK WINDOW 2
         BE    DEL$1
         CLC   EPGFILE2,=CL8'$0'
         BNE   INFO
         SPACE ,
* 5) CLEAR LOGGING FILE **********************************************
         SPACE ,
DEL$1    LH    R9,=H'256'
         STH   R9,EAMPREC              RECORD BUFFER
         MVC   EAMFILE,=CL8'$1'        FILE VARIABLE
         MVI   EAMMODB,EAMMOVM         INDEX PARAMETERS IN MOVE MODE
         BAL   R11,DELCALL
         SPACE ,
* 6) HANDLE THE RECORD MARKS IN $0 ***********************************
```

```
         SPACE ,
GTM$0    MVI   EAMMODB,EAMLOCM        PARAMETERS IN LOCATE MODE
         MVC   EAMFILE,=CL8'$0'       FILE VARIABLE
         L     R9,ABSOLUT
         ST    R9,EAMDISP             READ 1ST RECORD : DISP = 0
         LA    R9,FIRST
         ST    R9,AGTMPOS                            A(POS) = A(FIRST)
GTMNEXT  BAL   R11,GTMCALL
         CLI   EGLSR1,EAMAC16
         BE    END                    FILE EMPTY:--->
         CLI   EGLSR1,EAMOK12
         BE    GET$0                  EOF REACHED:--->
         SPACE ,
         L     R9,AGTMKEY
         ST    R9,AGTMPOS             NEW POSITION FOR GTM
         ST    R9,APTMKEY             INDEX FOR PUT
         L     R9,NEXT
         ST    R9,EAMDISP             SWITCH TO RELATIVE ACCESS (+1)
         MVC   EAMMARK,=XL2'0000'     DELETE MARKS
         BAL   R11,PTMCALL
         B     GTMNEXT                NEXT RECORD:--->
         SPACE 2
* 7) CHECK THE RECORD LENGTHS IN $0 **********************************
         SPACE ,
GET$0    MVI   EAMMODB,EAMMOVM        PARAMETERS IN MOVE MODE
         L     R9,ABSOLUT
         ST    R9,EAMDISP             READ 1ST RECORD: DISP = 0
         MVC   GETPOS,FIRST                             POS = FIRST
GETNEXT  MVC   EAMFILE,=CL8'$0'
         BAL   R11,GETCALL
         CLI   EGLSR1,EAMAC16
         BE    END                    FILE EMPTY:--->
         CLI   EGLSR1,EAMOK12
         BE    REN$1                  EOF REACHED:--->
         SPACE ,
         MVC   GETPOS,GETKEY          NEW POSITION FOR GET
         L     R9,NEXT
         ST    R9,EAMDISP             SWITCH TO RELATIVE ACCESS (+1)
         LH    R9,EAMLREC
         CH    R9,MAX                 COMPARE LENGTHS
         BNH   GETNEXT                RECORD VALID:--->
         SPACE ,
         OI    FLAG,X'FF'             DISPLAY ERROR
         MVI   EAMMARK2,EAMMK14       SET MARK 14
         BAL   R11,PTMCALL2
         SPACE ,
         MVI   EAMMARK2,EAMMK15       SET MARK 15
         MVC   EAMFILE,=CL8'$1'       COPY RECORD TO $1
         LH    R9,EAMLREC
         AH    R9,=H'9'
         STH   R9,EAMLREC                INDEX IN RECORD FIELD
         BAL   R11,PUTCALL
         B     GETNEXT                   NEXT RECORD:--->
         SPACE ,
* 8) RENUMBER THE LINES **********************************************
         SPACE ,
REN$1    L     R4,=F'99999999'           LAST INDEX
         MVI   EAMMODB,EAMMOVM           PARAMETERS IN MOVE MODE
         MVC   EAMFILE,=CL8'$1'          FILE VARIABLE
         L     R9,ABSOLUT                READ LAST RECORD:
         ST    R9,EAMDISP                                 DISP = 0
         MVC   GETPOS,LAST                                POS = LAST
GETNEXT2 BAL   R11,GETCALL
         CLI   EGLSR1,EAMAC16
         BE    END                       FILE EMPTY:--->
         CLI   EGLSR1,EAMOK08
         BE    END                       EOF REACHED:--->
         SPACE ,
         MVC   GETPOS,GETKEY             NEW POSITION FOR GET
         L     R9,PRE
         ST    R9,EAMDISP                RELATIVE ACCESS (-1)
         MVC   RENOLD,GETKEY
         CVD   R4,DW
         UNPK  RENNEW(8),DW(8)
         OI    RENNEW+7,X'F0'
         SPACE ,
         BAL   R11,RENCALL
         BCT   R4,GETNEXT2               NEXT RECORD:--->
         SPACE ,
* 9) CORRECTION LOOP (QUERY THE END CONDITION) ***********************
         SPACE ,
END      CLI   FLAG,X'00'
         BE    END2                      NO ERRORS:--->
         SPACE ,
         LA    R9,CMDPL2
         ST    R9,ACMDPL                 PARAMETERS FOR CORRECTION DIALOG
         B     DIALOG
         SPACE 2
END2     CLI   AKTHALT,EUPOK04
         BE    END3                      TERMINATE PROGRAM:--->
         SPACE ,
         LA    R9,CMDPL1
         ST    R9,ACMDPL                 PARAMETERS FOR DIALOG
         B     DIALOG
         SPACE ,
INFO     LA    R9,CMDPL3
```

```
         ST    R9,ACMDPL               PARAMETERS FOR DIALOG
         B     DIALOG
         SPACE 2
END3     WROUT NORMEND,STOP,MODE=LINE
         B     STOP
         SPACE ,
* 10) SUBROUTINES FOR EDT CALLS AND ERROR RECOVERY *******************
         SPACE ,
INFCALL  MVC   FUNCTION,=CL7'IEDTINF'  ERROR ANALYSIS
         LA    R1,INFPL
         L     R15,=V(IEDTINF)
         BALR  R14,R15
         B     ERRORUP
         SPACE ,
CMDCALL  MVC   FUNCTION,=CL7'IEDTCMD'  ERROR ANALYSIS
         L     R1,ACMDPL
         L     R15,=V(IEDTCMD)
         BALR  R14,R15
         B     ERRORUP
         SPACE ,
DELCALL  MVC   FUNCTION,=CL7'IEDTDEL'  ERROR ANALYSIS
         LA    R1,DELPL
         L     R15,=V(IEDTDEL)
         BALR  R14,R15
         B     ERRORAM
         SPACE ,
GTMCALL  MVC   FUNCTION,=CL7'IEDTGTM'  ERROR ANALYSIS
         LA    R1,GTMPL
         L     R15,=V(IEDTGTM)
         BALR  R14,R15
         B     ERRORAM
         SPACE ,
PTMCALL  MVC   FUNCTION,=CL7'IEDTPTM'  ERROR ANALYSIS
         LA    R1,PTMPL
         L     R15,=V(IEDTPTM)
         BALR  R14,R15
         B     ERRORAM
         SPACE ,
PTMCALL2 MVC   FUNCTION,=CL7'IEDTPTM'  ERROR ANALYSIS
         LA    R1,PTMPL2
         L     R15,=V(IEDTPTM)
         BALR  R14,R15
         B     ERRORAM
         SPACE ,
GETCALL  MVC   FUNCTION,=CL7'IEDTGET'  ERROR ANALYSIS
         LA    R1,GETPL
         L     R15,=V(IEDTGET)
         BALR  R14,R15
         B     ERRORAM
PUTCALL  MVC   FUNCTION,=CL7'IEDTPUT'  ERROR ANALYSIS
         LA    R1,PUTPL
         L     R15,=V(IEDTPUT)
         BALR  R14,R15
         B     ERRORAM
         SPACE ,
RENCALL  MVC   FUNCTION,=CL7'IEDTREN'  ERROR ANALYSIS
         LA    R1,RENPL
         L     R15,=V(IEDTREN)
         BALR  R14,R15
         B     ERRORAM
GETGCALL MVC   FUNCTION,=CL7'IEDTGET'  ERROR ANALYSIS
         LA    R1,GETGPL
         L     R15,=V(IEDTGET)
         BALR  R14,R15
         B     ERRORAM
         SPACE ,
*-- ERROR RECOVERY : IEDTINF / IEDTCMD ------------------------------*
ERRORUP  CLC   EGLMRET,=AL2(EUPRETOK)
         BNE   ERROR
         BR    R11
*-- ERROR RECOVERY : IEDTGET/IEDTGTM/IEDTPUT/IEDTPTM/IEDTREN/IEDTDEL
ERRORAM  CLC   EGLMRET,=AL2(EAMACERR)
         BNE   ERRAM02
         CLI   EGLSR1,EAMAC16
         BRE   R11                     FILE EMPTY: --->
         CLI   EGLSR1,EAMAC20
         BE    GET$0                   NO MARKS IN FILE: --->
         SPACE ,
ERRAM02  CLC   EGLMRET,=AL2(EAMRETOK)
         BRE   R11                     NO ERRORS: --->
         SPACE ,
*-- ERROR RECOVERY : REMAINING ERRORS --------------------------------*
ERROR    LH    R9,EGLRMSGL
         LTR   R9,R9
         BZ    ERROR2                  NO MESSAGE: --->
         SPACE ,
         AH    R9,=H'5'
         STH   R9,VARMLD
         MVC   VARMLDF,EGLRMSGF
ERROR2   UNPK  MAINCODE(5),EGLMRET(3)  IGNORE LAST BYTE
         NC    MAINCODE,=X'0F0F0F0F'
         MVI   MAINCODE+4,C''''
         TR    MAINCODE(4),DRUCKTB
         UNPK  SUBCODE(3),EGLSR1(2)    IGNORE LAST BYTE
         NC    SUBCODE,=X'0F0F'
         MVI   SUBCODE+2,C''''
         TR    SUBCODE(2),DRUCKTB
         WROUT ERRMLD,STOP,MODE=LINE
         WROUT VARMLD,STOP,MODE=LINE
         B     STOP
         SPACE ,
STOP     TERM  ,
         SPACE ,
* DATA AREA *******************************************************
         SPACE ,
*-- EDTGLCB, EDTUPCB, EDTAMCB, EDTPARG-------------------------------*
         IEDTGLCB C
         IEDTUPCB C
         IEDTAMCB C
         IEDTPARG C
*-- COMMAND SEQUENCE FOR DIALOG -------------------------------------*
CMDDIA   DC    Y(CMDDIAL)
         DC    CL2' '
         DC    C'SETF(0);PAR SPLIT=OFF,LOWER=ON,SCALE=ON,INDEX=ON;DIALOG'
CMDDIAL  EQU   *-CMDDIA
*-- COMMAND SEQUENCE FOR CORRECTION DIALOG --------------------------*
CMDKOR   DC    Y(CMDKORL)
         DC    CL2' '
         DC    C'SETF(0);PAR GLOBAL,SPLIT=12 $1,LOWER=ON,SCALE=ON,INDEX=ON'
         DC    C',PROTECTION=ON;DIALOG'
CMDKORL  EQU   *-CMDKOR
*-- COMMAND SEQUENCE FOR DIALOG -------------------------------------*
CMDINF   DC    Y(CMDINFL)
         DC    CL2' '
         DC    C'DIALOG'
CMDINFL  EQU   *-CMDINF
*-- MESSAGE1 FOR DIALOG ---------------------------------------------*
MLD1DIA  DC    Y(MLD1DIAL)
         DC    CL2' '
VERSION  DS    CL12
         DC    C' '
         DC    C'TESTUP: DATA ENTRY DIALOG (END USING HALT/RETURN/K1 *
               )'
MLD1DIAL EQU   *-MLD1DIA
*-- MESSAGE 2 FOR DIALOG (DUMMY MESSAGE) ----------------------------*
MLD2DIA  DC    Y(MLD2DIAL)
         DC    CL2' '
MLD2DIAL EQU   *-MLD2DIA
*-- MESSAGE1 FOR CORRECTION DIALOG ----------------------------------*
MLD1KOR  DC    Y(MLD1KORL)
         DC    CL2' '
         DC    C'TESTUP: CORRECTION DIALOG (END USING HALT/RETURN/K1)'
MLD1KORL EQU   *-MLD1KOR
*-- MESSAGE2 FOR CORRECTION DIALOG ----------------------------------*
```

```
MLD2KOR  DC    Y(MLD2KORL)
         DC    CL2' '
         DC    C'TESTUP: ERROR LOG'
MLD2KORL EQU   *-MLD2KOR
*-- MESSAGE, IF THE USER HAS SWITCHED -------------------------------*
MLD1INF  DC    Y(MLD1INFL)
         DC    CL2' '
         DC    C'TESTUP: CHECK THE DATA IN WORK FILE 0 ONLY'
MLD1INFL EQU   *-MLD1INF
*-- INDEX FIELDS, RECORD FIELDS -------------------------------------*
FIRST    DC    XL8'0000000000000000'
LAST     DC    XL8'FFFFFFFFFFFFFFFF'
RENOLD   DS    CL8
RENNEW   DS    CL8
GETPOS   DS    CL8
PUTREC   EQU   *
PUTKEY   EQU   *
GETKEY   DS    CL8
BLANK    DC    C' '
GETREC   DS    CL256
*-- ADDRESS FIELDS FOR LOCATE MODE ----------------------------------*
APTMKEY  DS    A
AGTMPOS  DS    A
AGTMKEY  DS    A
AGTMREC  DS    A
*-- PARAMETER LIST FOR CMD CALL VARIABLE ----------------------------*
ACMDPL   DC    A(CMDPL1)
*-- PARAMETER LISTS FOR EDT CALLS -----------------------------------*
INFPL    DC    A(EDTGLCB)
CMDPL1   DC    A(EDTGLCB)
         DC    A(EDTUPCB)
         DC    A(CMDDIA)
         DC    A(MLD1DIA)
         DC    A(MLD2DIA)
CMDPL2   DC    A(EDTGLCB)
         DC    A(EDTUPCB)
         DC    A(CMDKOR)
         DC    A(MLD1KOR)
         DC    A(MLD2KOR)
CMDPL3   DC    A(EDTGLCB)
         DC    A(EDTUPCB)
         DC    A(CMDINF)
         DC    A(MLD1INF)
         DC    A(MLD2DIA)
GETPL    DC    A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(GETPOS)
         DC    A(GETKEY)
         DC    A(GETREC)
PUTPL    DC    A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(PUTKEY)
         DC    A(PUTREC)
GTMPL    DC    A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(AGTMPOS)
         DC    A(AGTMKEY)
         DC    A(AGTMREC)
PTMPL    DC    A(EDTGLCB)
         DC    A(EDTAMCB)           LOCATE MODE
         DC    A(APTMKEY)
PTMPL2   DC    A(EDTGLCB)
         DC    A(EDTAMCB)           MOVE MODE
         DC    A(PUTKEY)
DELPL    DC    A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(FIRST)
         DC    A(LAST)
RENPL    DC    A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(RENOLD)
         DC    A(RENNEW)
GETGPL   DC    A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(0)
         DC    A(0)
         DC    A(EDTPARG)
*-- MESSAGES FOR WROUT ---------------------------------------------*
NORMEND  DC    Y(NORMENDL)
         DC    CL3' '
         DC    C'TESTUP TERMINATED'
NORMENDL EQU   *-NORMEND
ERRMLD   DC    Y(ERRMLDL)
         DC    CL3' '
         DC    C'EDT ERROR: FUNCTION ='
FUNKTION DS    CL7
         DC    C', MAINCODE = X'''
MAINCODE DS    CL5
         DC    C', SUBCODE = X'''
SUBCODE  DS    CL3
ERRMLDL  EQU   *-ERRMLD
VARMLD   DC    Y(VARMLDL)
         DC    CL3' '
VARMLDF  DC    CL80'NO ERROR MESSAGE TEXT (EGLMSGF) EXISTS'
VARMLDL  EQU   *-VARMLD
         SPACE ,
*-- AUXILIARY DATA ---------------------------------------------------*
FLAG     DS    X
AKTHALT  DS    CL1
DW       DS    D
DRUCKTB  DC    CL16'0123456789ABCDEF'
MAX      DC    H'40'
ABSOLUT  DC    F'0'
NEXT     DC    F'1'
PRE      DC    F'-1'
SAVEAREA DS    18F
* END
         SPACE ,
         END   TESTUP
```

3.6.3 EDT as a subroutine of a C program

```
#include <stdio.h>
#include "iedglcb.h"
#include "iedupcb.h"

typedef struct vrec { unsigned short length;
                      short res1;
                      char text[257];        } VREC;

 main()
{
       struct IEDGLCB_mdl iedtglcb=IEDGLCB_INIT;
       struct IEDUPCB_mdl iedtupcb=IEDUPCB_INIT;

       VREC command;
       VREC message1;
       VREC message2;
       char message [81];

  extern void iedtcmd( );

       void fill_vrec (VREC *,const char *);

  /* Prevent switch to line mode */
      IEDUPCB_SET_NO_INHIBIT(iedtupcb);
       iedtupcb.inhibit.bit.no_edit = 1
       iedtupcb.inhibit.byte |= iedtupcb.inhibit.bit.no_edit_only = 1;

 /* Define statement sequence and message lines */
       fill_vrec (&command,"PAR GL, LOWER=ON,CODE=ISO;DIALOG;HALT");
       fill_vrec (&message1,"DIALOG-ENDE MIT HALT ODER <K1>");
       fill_vrec (&message2,"");

       printf ("EDT starts with the settings CODE=ISO and LOWER=ON\n");

/* Call EDTs */
       iedtcmd(&iedtglcb,&iedtupcb,&command,&message1,&message2);

 /* Output return code*/
       printf (“EDT terminates with return code %X \n“,
                iedtglcb.IEDGLCB_RC_NBR);
       if (iedtglcb.return_message.structured_msg.rmsgl > 0 )

          {/* Message available */
           strncpy (message,
                    iedtglcb.return_message.structured_msg.rmsgf,
                    iedtglcb.return_message.structured_msg.rmsgl);
           message[iedtglcb.return_message.structured_msg.rmsgl+1]=0x00;
           printf (“Message text: %s“,message);
          }
}
```

```
 void fill_vrec ( VREC * p, char * textp)
{
       int l_text;
       p->res1 = 0x4040;
       if ((l_text = strlen(textp)) > 256 )
          { l_text = 256; };
       strncpy (p->text,textp,l_text);
       p->length =  l_text + 4;
}
```

The C program is bound to EDT by means of the following procedure:

```
/BEGIN-PROC
/START-PROGRAM FROM=$BINDER
//START-LLM-CREATION INT-NAME=CEDT
//INCLUDE-MODULE LIB=TESTLIB,ELEM=CEDT#
//INCLUDE-MODULE LIB=TESTLIB,ELEM=CEDT@
//INCLUDE-MODULE LIB=$.SYSLNK.EDT.166,ELEM=IEDTGLE
//RESOLVE-BY-AUTOLINK LIB=$.SYSLNK.CRTE.020
//MOD-SYM-VIS *ALL,VIS=NO
//SAVE-LLM LIB=TESTLIB,ELEM=CEDT
//END
/END-PROCEDURE
```

Then you use the START-PROGRAM command to call the program:

```
/START-PROG *MOD(TESTLIB,CEDT),RUN-MOD=ADVANCED
```

```
 % BLS0523 ELEMENT,CEDT‘,VERSION,,FROM LIBRARY,:J:$EDT.TESTLIB‘ IN
PROCESSING
%  BLS0524 LLM,CEDT‘,VERSION,,OF,1996-02-07:14:18:11‘ LOADED
```

EDT starts with the settings CODE=ISO and LOWER=ON

```
   1.00 ................ ...........................................
   2.00 ............................................................
   3.00 .................................. ..........................
   4.00 ............................................................




lDIALOG END WITH HALT OR <K1>...............................0001.00:001(0)
```

```
 EDT terminates with return code 40000.
```

Connection to an L mode application

If the L mode subroutine interface (see chapter “L mode subroutine interfaces” on page 137ff) is used, it is possible to use the current data area in the first call via the IEDTGLE interface. This permits parallel use of the two program interfaces.

Before the first call via the IEDTGLE interface, control block EDTGLCB must contain the indicator byte EGLINBD with the bit settings EGLINIT=X’02’ and EGLDTVD=X’02’, and the address of the EDT data area (word 2 of the L mode parameter list) must be entered in EGLDATA.

This informs EDT that EGLDATA is valid and has already been initialized. After the first call, EGLDATA (EDTGLCB) must not be changed.

If a parallel EDT call to the L mode subroutine interface (LU-V15) is issued via the IEDTGLE module, statements via the L mode interface will be accepted like in L mode dialog from that point in time. It is then possible, for example, to process job variables by means of @GETJV.

Exception: user statements are not possible.

4 External statement routines: @USE

External statements

EDT permits the user to create his/her own statements.

To this end, the desired function of the new statement must be implemented as an external user statement routine, which must be stored as a module in an object module library.

The routine is defined by means of @USE (see the description in the "EDT Statements" manual [1]) and can then be called as a subroutine with the aid of the user escape symbol, which is also defined by means of @USE (=external statement).

The external statement may be followed by a text, which is then passed to the routine. This text can be processed by the statement routine and a message can be returned to EDT when the routine is terminated.

Special application as a statement filter

When EDT is called via the IEDTCMD function, the calling program can define a user statement routine with an "empty" escape symbol (see also section “IEDTCMD - Execute EDT statements” on page 18ff).

@USE COMMAND = ' ' [($\left\{\begin{array}{l}\text{entry}\\ *\end{array}\right\}$ [,modlib])]

This routine then receives every statement which is entered

– in the statement line in the F mode dialog or
– in the L mode dialog.

The routine does not receive

– the statement codes entered in F mode dialog or

– any statements which are entered via the program interfaces IEDTCMD and IEDTEXE or via the L mode interface (see chapter “L mode subroutine interfaces” on page 137ff).

The statement received from EDT cannot be modified in the statement routine, but it is possible to place the following return codes in field EGLSR1 before control is returned to EDT:

EUPOK00 ...	The statement is to be executed.
EUPOK12 ...	EDT is to return to the calling program (only in the case of the statement filter).
EUPOK24 ...	The statement is not to be executed.

A message can also be passed to EDT at the same time.

EDT passes the following parameters:

– **EDTGLCB**
  The global EDT control block (see section "EDTGLCB - Global control block" on page 53ff). The EDT environment has already been initialized (indicated by the flags EGLINIT and EGLDTVD and address EGLDATA in the control block).

– **COMMAND**
  The text which was entered when the external statement was input. The escape symbol is not passed. The maximum length of this text is 256 bytes plus a length field of 4 bytes. If the statement routine is used as a statement filter, it should be noted that statements longer than 256 bytes will be truncated on the right.

Overview table

Control blocks: see section "EDTGLCB - Global control block" on page 53ff.

Parameter list	:	A(EDTGLCB, COMMAND)

Call parameters		**Return parameters**	
EDTGLCB:	EGLUNIT EGLVERS EGLDATA EGLENTRY EGLINDB EGLUSR1 EGLUSR2 EGLUSR3	EDTGLCB:	EGLRETC EGLRMSG EGLCMDS EGLFILE EGLUSR2
COMMAND			

Return codes for external statement routines

EGLMRET	**EGLSR1**
EUPRETOK	00
	EUPOK12
	EUPOK24
EUPSYERR	00
EUPRTERR	00
EUPUSERR	00
EUPPAERR	EUPPA04
	EUPPA12

For the meanings of the return codes see section “EDTGLCB - Global control block” on page 53ff.

The following functions are available to the statement routine for processing:

– the record access functions IEDTPUT, IEDTPTM, etc. (see section “Logical record access functions” on page 27ff)

– the function IEDTEXE for executing an EDT statement (see section “IEDTEXE - Execute EDT statements without screen dialog” on page 24ff).
The following statements must not be specified with IEDTEXE:
  – a further user statement,
  – the statements @DIALOG, @EDIT, @END, @HALT and @RETURN, and
  – the procedure calls @DO and @INPUT.

Entering an external statement

Input of the user escape symbol defined for an external statement routine causes this routine to be loaded and started.

Any desired text can be passed to the statement routine.

Statement	Operands	F mode / L mode
<usersymb>	({ entry [text] | [text] })	

usersymb	The user escape symbol defined for the statement routine by means of @USE.
entry	The entry name of the statement routine. If the entry name was defined by means of @USE, the entire string following the escape symbol is passed as a parameter to the external routine (entry is not evaluated). If an asterisk (*) was specified as the entry name in @USE, EDT regards the characters following the escape symbol, up to the first blank or to the end of the statement, as the entry name (a BIND call is issued only when the statement is entered). The text after the entry name is passed as a parameter.
text	Any desired text, which is passed to the statement routine defined by means of @USE. In F mode, the text is terminated by the statement delimiter ";" (semicolon), otherwise it is terminated by the end of the statement. The text may be up to 256 bytes long. If @LOWER ON is active, any characters in "text" which are enclosed within single quotes or the currently defined quote characters are not converted to uppercase letters.

The EDT statement symbol (see the description of @ in the "EDT Statements" manual [1]) must not be specified before external statements.

If the user escape symbol is the same as the current statement symbol, the EDT statement has the higher priority. It is the user’s responsibility to avoid conflicts between the statements.

Example 1

Entry defined in @USE

```
Entryname      : JOBVAR

Syntax         : CATJV name
                 ERAJV name
                 GETJV name, zlnr
                 SETJV name, zlnr
Deklaration    : @USE COMMAND = '*' (JOBVAR,PRIVLIB)

Anwendung      : *CATJV JV.TEST   ==> Calls module JOBVAR with the
                                      parameter 'CATJV JV.TEST'

                 *SETJV JV.TEST,3 ==> Calls module JOBVAR with the
                                      parameter 'SETJV JV.TEST,3'
```

Example 2

Entry defined by external statement

```
                 HELP <msg7id>

Declaration    : @USE COMMAND = '*' (*,PRIVLIB)

Application    : *SORT 20-100     ==> Calls module SORT with the
                                      parameter '20-100'

                 *HELP EDT5100    ==> Calls module HELP with the
                                      parameter 'EDT5100'
```

```
**********************************************************************
* SEL : EXAMPLE OF AN EXTERNAL USER ROUTINE ('USE' ROUTINE)          *
*                                                                    *
**********************************************************************
*  FUNCTION :  WITH *SEL L=<LIB>, A TABLE OF CONTENTS OF THE
*                     PLAM LIBRARY <LIB> IS OUTPUT TO WORK FILE 0.
*              THEN THE USER CAN SPECIFY THE WORK FILE IN WHICH
*                     THE SELECTED ELEMENT IS TO BE OPENED,
*                     USING MARKS 1 THROUGH 9 [F3].
*              *SEL IS USED TO SELECT AN ELEMENT.
*              *SEL N IS USED TO DISPLAY THE NAME OF THE
*                     ELEMENT IN THE MESSAGE LINE.
* 1)  INITIALIZE CONTROL BLOCKS
* 2)  SYNTAX CHECK
* 3)  PROCESSING OF THE STATEMENT '*SEL L=<LIBRARY>'
* 4)    IEDTEXE: DEFINE STANDARD LIBRARY
* 5)    IEDTEXE: OUTPUT TABLE OF CONTENTS
* 6)  PROCESSING OF THE '*SEL' STATEMENT
* 7)    IEDTGTM: FETCH MARKED RECORDS
* 8)    IEDTEXE: OPEN ELEMENT IN SPECIFIED WORK FILE
* 9)    IEDTPTM: DELETE PROCESSED MARK
* 10) PROCESSING OF THE '*SEL N' STATEMENT
* 11)   IEDTGET: FETCH LOCAL FILE DESCRIPTION
* 12) RETURN TO EDT
*
*  ENTRY POINT:  SEL
**********************************************************************
SEL      START
SEL      AMODE
SEL      RMODE ANY
         GPARMOD 31
**********************************************************************
*      DEFINITION SECTION
**********************************************************************
         MACRO
&NAME    VREC  &TEXT
&NAME    DS    OH
         DC    Y(&NAME.E-&NAME)    LENGTH FIELD
         DC    X'4040'
         DC    C&TEXT              TEXT
&NAME.E  EQU   *
         MEND
         PRINT NOGEN
         REGS                      EQUATES FOR REGISTERS
         SPACE ,
EDTGLCB  IEDTGLCB  D
         SPACE ,
SHOWLINE DSECT
         DS    CL1
RPOST    DS    CL1                 TYPE
         DS    CL4
RPOSE    DS    CL32                ELEMENT
         DS    CL3
RPOSV    DS    CL12                VERSION
         DS    CL10
         SPACE ,
**********************************************************************
*      STATEMENT SECTION
**********************************************************************
* INTERFACES:
*   1ST PARAMETER: A(EDTGLCB)
*   2ND PARAMETER: A(STATEMENT TEXT)
**********************************************************************
SEL      CSECT
SEL      AMODE ANY
SEL      RMODE ANY
         GPARMOD 31
         USING SEL,R15
         STM   R14,R12,12(R13)      SAVE REGISTERS
         ST    R13,SAVR13
         LA    R13,SAVEAREA         SAVE AREA FOR EDT CALLS
         DROP  R15
*
         LR    R11,R15              LOAD BASE REGISTERS
         USING SEL,R11
         L     R10,0(R1)            1ST PARAMETER
         L     R9,4(R1)             2ND PARAMETER
         USING EDTGLCB,R10
         LA    R1,4(R9)             R1: START OF TEXT
         LH    R2,0(R9)             R2: LENGTH OF TEXT
         SH    R2,=H'4'             SUBTRACT LENGTH OF HEADER
         SPACE ,
* 1)  INITIALIZE CONTROL BLOCKS ***************************************
         SPACE ,
         ST    R10,GTMPL            TRANSFER A(EDTGLCB) FROM ENTRY POINT
         ST    R10,PTMPL
         ST    R10,CMDPL
         ST    R10,PARPL
         LH    R3,=H'8'             SUPPLY KEY LENGTH
         STH   R3,EAMLKEY1          INPUT KEY
         STH   R3,EAMPKEY           OUTPUT KEY
         SPACE ,
* 2)  SYNTAX CHECK ****************************************************
         SPACE ,
         LTR   R2,R2                CHECK PARAMETER LENGTH
         BZ    DOGETM               NO PARAMETERS -->
         BAL   R14,NXTCHAR          SKIP BLANKS
         B     DOGETM
         CH    R2,HKEYL
         BL    DOGETO
         CLC   0(3,R1),KEYWORD      KEYWORD 'SEL'?
         BNE   DOGETO               NO -->
         AH    R1,HKEYL             SKIP KEYWORD
         SH    R2,HKEYL
         LTR   R2,R2
         BZ    DOGETM               --> PROCESS '*SEL'
         CLI   0(R1),' '            BLANK AS DELIMITER
         BNE   ERRSY                OTHERWISE SYNTAX ERROR
         SPACE ,
DOGETO   BAL   R14,NXTCHAR          SKIP BLANKS
         B     DOGETM               NO FURTHER PARAMETERS -->
         CLI   0(R1),'N'            KEYWORD 'N'?
         BE    DOGETN               --> PROCESS '*SEL N'
         CLI   0(R1),'L'           KEYWORD 'L'?
         BNE   ERRSY               NO, SYNTAX ERROR
         SPACE
         BAL   R14,NXTCHAR1
         BNE   ERRSY               'L' ONLY (WITHOUT NAME), SYNTAX ERROR
         CLI   0(R1),'='
         BNE   ERRSY               SYNTAX ERROR
         BAL   R14,NXTCHAR1
         B     ERRSY               NO LIBRARY NAME, SYNTAX ERROR
         B     DOGETL              --> PROCESS '*SEL L=<LIB>'
         SPACE 2
NXTCHAR  CLI   0(R1),C' '          BLANKS ?
         BNE   4(0,R14)            NO -->
NXTCHAR1 LA    R1,1(0,R1)          (R1)=(R1)+1
         BCT   R2,NXTCHAR          (R2)=(R2)-1
         BR    R14                 END OF INPUT -->
         EJECT
* 3)  PROCESS THE STATEMENT'*SEL L=<LIBRARY>' ************************
         SPACE ,
DOGETL   MVC   CMDSHOWL(54), DELETE BLANKLIB FIELD
         LA    R3,CMDPAR
         ST    R3,ACMD             ENTER ADDRESS OF STATEMENT
         LR    R4,R1
         BAL   R14,GETLNG
         BCTR  R2,0                LENGTH FOR MVC
         EX    R2,MVCLIB           ENTER <LIB> IN CMDSHOWL
         SPACE ,
* 4)    IEDTEXE: DEFINE STANDARD LIBRARY ******************************
         SPACE ,
         MVC   CMDPARL(54),CMDSHOWL
         LA    R1,CMDPL
         L     R15,=V(IEDTEXE)     EXECUTE @PAR STATEMENT
         BALR  R14,R15
         CLC   EGLMRET,NOERR
         BNE   ERRCMD
         SPACE ,
* 5)    IEDTEXE: OUTPUT TABLE OF CONTENTS *****************************
         SPACE ,
         MVC   STDLIB(54),CMDSHOWL SAVE LIBRARY NAME
         LA    R3,CMDSHOW
         ST    R3,ACMD             ENTER ADDRESS OF STATEMENT
         LA    R1,CMDPL
         L     R15,=V(IEDTEXE)     EXECUTE @SHOW STATEMENT
         BALR  R14,R15
         CLC   EGLMRET,NOERR
         BNE   ERRCMD
         B     MLD1                --> SPECIFY MESSAGE
         SPACE ,
MVCLIB   MVC   CMDSHOWL(*-*),0(R4)
         EJECT
* 6)  PROCESS THE '*SEL' STATEMENT *************************************
         SPACE ,
DOGETM   CLC   STDLIB(54),BLANKLIB '*SEL L=<LIB>' QUERIED BEFOREHAND?
         BNE   LIBMOVED            YES -->
         BAL   R14,GETFILD         FETCH LOCAL FILE DESCRIPTION
         CLC   EPLOPNFL(54),BLANKLIB ELEMENT ALREADY OPENED?
         BNE   MLD3                YES --> ERROR MESSAGE
         CLC   EPLSTDL(54),BLANKLIB STANDARD LIBRARY DEFINED?
         BE    MLD2                NO --> REQUEST 'SEL L=..'
         MVC   STDLIB(54),EPLSTDL  SAVE LIBRARY NAME
LIBMOVED XC    CNT0,CNT0           SET COUNTER TO ZERO
         SPACE ,
* 7)    IEDTGTM: FETCH MARKED RECORDS **********************************
         SPACE ,
         MVI   EAMMODB,EAMLOCM     LOCATE MODE
         MVC   EAMFILE,EGLFILE     ENTER CURRENT WORK FILE
         XC    EAMDISP,EAMDISP     READ 1ST RECORD: DISP=0
         LA    R9,ZERO
         ST    R9,AGTMPOS                        A(POS)=0
GETNEXT  LA    R1,GTMPL
         L     R15,=V(IEDTGTM)     FETCH ADDRESS OF MARKED RECORD
         BALR  R14,R15
         CLC   EGLMRET,NOERR
         BNE   ERREAM
         CLI   EGLSR1,EAMOK12      EOF REACHED?
         BE    ENDE                --> TERMINATE PROCESSING
         SPACE ,
*-- TRANSFER ELEMENT NAME + VERSION + TYPE -------------------------
         USING SHOWLINE,R9
         L     R9,AGTMREC
         MVC   ELEMPOS(64),RPOSE   TRANSFER ELEMENT NAME
         LA    R2,64
         LA    R1,ELEMPOS
         BAL   R14,GETLNG          DEFINE LENGTH
         CLI   RPOSV,'@'           VERSION NOT SPECIFIED -->
         BE    NOVER
         MVI   0(R1),'('
         LA    R1,1(R1)
         MVC   0(24,R1),RPOSV      TRANSFER VERSION
         LA    R2,24
         BAL   R14,GETLNG          DEFINE LENGTH
         MVI   0(R1),')'
NOVER    MVI   1(R1),','
         MVC   2(1,R1),RPOST       TRANSFER TYPE
         LA    R1,3(R1)            POSITION BEHIND
         BAL   R14,GETMARK         ENTER MARK
         SPACE ,
* 8)    IEDTEXE: OPEN ELEMENT IN SPECIFIED WORK FILE ******************
         SPACE ,
         LA    R2,CMDOPEN          ENTER ADDRESS OF STATEMENT
         ST    R2,ACMD
         SR    R1,R2
         STH   R1,CMDOPEN          ENTER LENGTH OF STATEMENT
         LA    R1,CMDPL
         L     R15,=V(IEDTEXE)
         BALR  R14,R15             EXECUTE @OPEN STATEMENT
         CLC   EGLMRET,NOERR
         BNE   ERRCMD
         SPACE ,
* 9)    IEDTPTM: DELETE PROCESSED MARK ********************************
         SPACE ,
         XC    EAMMARK(2),EAMMARK  DELETE MARK FIELD
         L     R9,AGTMKEY
         ST    R9,AGTMPOS          TRANSFER ADDRESS OF RECORD
         LA    R1,PTMPL
         L     R15,=V(IEDTPTM)
         BALR  R14,R15
         CLC   EGLMRET,NOERR
         BNE   ERREAM
         SPACE ,
         MVC   EAMDISP,NEXT        SWITCH TO RELATIVE ACCESS (+1)
         MVI   ELEMPOS,' '         DELETE ELEMENT NAMES
         MVC   ELEMPOS+1(L'ELEMPOS-1),ELEMPOS
         LH    R8,CNTO
         AH    R8,=H'1'            COUNT ELEMENT
         STH   R8,CNTO
         B     GETNEXT             --> NEXT RECORD
         SPACE ,
*-- SPECIFY END MESSAGE ------------------------------------------*
ENDE     CLI   CNT,0               NUMBER OF @OPENS PERFORMED?
         BE    MLD1                NONE --> MESSAGE
         OI    CNT,X'F0'           RENDER PRINTABLE
         MVC   MSGOK+4(1),CNT      ENTER NUMBER IN MESSAGE
         LA    R1,MSGOK            ADDRESS MESSAGE
         B     RETURN              --> RETURN TO EDT
         EJECT
* 10) PROCESS THE '*SEL N' STATEMENT *********************************
         SPACE ,
DOGETN   MVI   MSGNAM,' '           DELETE NAME FIELD
         MVC   MSGNAM+1(63),MSGNAM
         BAL   R14,GETFILD          FETCH LOCAL FILE DESCRIPTION
         CLI   EPLOPEN,'P'          PLAM ELEMENT OPENED?
         BNE   MLD5                 NO --> ERROR MESSAGE
         MVC   MSGNAM(64),EPLOPNE   TRANSFER ELEMENT NAME
         LA    R1,MSGNAM
         LA    R2,64
         BAL   R14,GETLNG           DEFINE LENGTH
         MVI   0(R1),','
         MVC   1(1,R1),EPLOPNT      TRANSFER TYPE
         LA    R1,MSGNM             ADDRESS MESSAGE
         B     RETURN
         SPACE ,
* 11)   IEDTGET: FETCH LOCAL FILE DESCRIPTION ************************
         SPACE ,
GETFILD  ST    R14,SAVR14          SAVE REGISTER 14
         MVI   EAMMODB,EAMMOVM     MOVE MODE
         MVC   EAMFILE,EGLFILE     CURRENT FILE NUMBER
         MVI   EAMFILE,'L'         LOCAL FILE DESCRIPTION
         MVC   EAMLREC(2),=AL2(EPLPARLL)
         MVC   EAMPREC(2),=AL2(EPLPARLL)  SUPPLY LENGTH FIELD VALUES
         XC    EAMDISP,EAMDISP
         LA    R1,PARPL
         L     R15,=V(IEDTGET)     REQUEST LOCAL FILE DESCRIPTION
         BALR  R14,R15
         CLC   EGLMRET,NOERR
         BNE   ERREAM
         L     R14,SAVR14
         BR    R14
         EJECT
*--  HELP ROUTINES ----------------------------------------------------*
GETLNG   EQU   *                   DETERMINE LENGTH OF STRING
         LR    R15,R2                                UP TO 1ST BLANK
GETN     CLI   0(R1),' '           BLANKS?
         BE    GETLEN1             YES -->
         LA    R1,1(R1)
         BCT   R15,GETN            CHECK NEXT CHARACTER
GETLEN1  SR    R2,R15              DEFINE LENGTH
         BR    R14                 --> RETURN
         SPACE ,
*--  MARK IS ENTERED AS FILE NUMBER IN @OPEN STATEMENT
GETMARK  EQU   *
         LA    R9,9                INITIALIZE LOOP
         LH    R8,EAMMARK
         CLI   EAMMARK2,EAMMK09    MARK 9?
         BE    STFILNR
         BCTR  R9,0
         CLI   EAMMARK2,EAMMK08    MARK 8?
         BE    STFILNR
         BCTR  R9,0
GETMARK0 EQU   *                   TEST EAMMARK1
         CLM   R8,1,=X'80'         MARK SET?
         BE    STFILNR             YES  -- >
         SLL   R8,1                  SHIFT TO THE LEFT
         BCT   R9,GETMARK0           CONTINUE CHECK
         B     MLD1                  NO VALID MARK --> ERROR
         SPACE ,
STFILNR  STC   R9,CMDFILNR           MAKE RETRIEVED MARK PRINTABLE
         OI    CMDFILNR,X'F0'        AS FILE NUMBER
         CLC   CMDFILNR(1),EGLFILE+1 IS CURRENT FILE?
         BE    MLD4                  YES  -- > ERROR
         BR    R14
         EJECT
* -- ERROR RECOVERY + MESSAGE SPECIFICATION ----------------------------*
         SPACE ,
ERREAM   CLI   EGLSR1,EAMAC16        FILE EMTPY -->
         BE    MLD2
         CLI   EGLSR1,EAMAC20        NO MARKS -->
         BE    ENDE
ERRCMD   EQU   *
         LH    R9,EGLRMSGL
         LTR   R9,R9                 EDT MESSAGE?
         BZ    ERRINT
         SH    R9,=H'10'             REMOVE '%  EDTNNNN'
         STH   R9,EGLRMSGL           CORRECT LENGTH
         BCTR  R9,0
         MVC   EGLRMSGF(L'EGLRMSGF-10),EGLRMSGF+10 SHIFT MESSAGE
         MVC   EGLMRET,=AL2(EUPSYERR)  RETURN WITH ERROR
         B     RETURN2
         SPACE 2
MLD1     LA    R1,MSG1               'MARK ELEMENT'
         B     RETURN
MLD2     LA    R1,MSG2               '*SEL L=<LIB> REQUIRED'
         B     RETURN
MLD3     LA    R1,MSG3               'OPEN NOT POSSIBLE'
         MVC   NMLD3F(1),CMDFILNR    ENTER FILE NUMBER
         B     RETURN
MLD4     LA    R1,MSG4               'INVALID MARK'
         B     RETURN
MLD5     LA    R1,MSG5               'NO PLAM ELEMENT'
         B     RETURN
ERRINT   LA    R1,MSGERR             'INTERNAL ERROR'
         B     ERR
         SPACE
ERR      EQU   *
         MVI   EGLSR1,0
         MVC   EGLMRET,=AL2(EUPSYERR)
         B     RETURN1
ERRSY    EQU   *
         LA    R1,MSGSY              SYNTAX ERROR
         MVI   EGLSR1,0
         MVC   EGLMRET,=AL2(EUPSYERR)
         B     RETURN1
         SPACE ,
* 12) RETURN TO EDT **************************************************
         SPACE ,
RETURN   MVI   EGLSR1,0            NO ERRORS
         MVC   EGLMRET,=AL2(EUPRETOK)
         SPACE ,
RETURN1  XR    R2,R2
         ICM   R2,3,0(R1)
         SH    R2,=H'4'            SUBTRACT LENGTH OF HEADER
         STH   R2,EGLRMSGL         ENTER LENGTH FIELD
         MVC   EGLRMSGF,4(R1)      TRANSFER TEXT
RETURN2  EQU   *
         L     R13,SAVR13
         LM    R14,R12,12(R13)
         BR    R14                 RETURN TO EDT
         EJECT
* DATA AREA **********************************************************
         LTORG
         SPACE ,
HKEYL    DC    H'3'
KEYWORD  DC    C'SEL '
*
MSGOK    DC    AL2(MSGOKE-MSGOK)
         DC    CL2'  '
         DC    C'X ELEMENTS OPENED: L= '
STDLIB   DC    CL54' '
MSGOKE   EQU   *
*
MSGNM    DC    AL2(MSGNME-MSGNM)
         DC    CL2'  '
         DC    C'E= '
MSGNAM   DC    CL66' '
MSGNME   EQU   *
*
MSGSY    VREC  'SYNTAX: SEL L=<LIBRARY>/[]/N'
MSG1     VREC  'PLEASE MARK ELEMENT WITH 1 [F3] - 9 [F3]'
MSG2     VREC  '*SEL L=<LIBRARY> REQUIRED'
MSG3     VREC  'ELEMENT IN $X OPENED, FUNCTION NOT POSSIBLE'
NMLD3F   EQU    MSG3+16
MSG4     VREC  'INVALID MARK'
MSG5     VREC  'NO PLAM ELEMENT OPENED'
MSGERR   VREC  'INTERNAL ERROR'
*
CMDOPEN  DS    H
         DC    CL2'  '
         DC    C'SETF (X);'
         DC    C'OPEN E='
ELEMPOS  DC    CL100' '
CMDFILNR EQU   CMDOPEN+10
*
         DS    0H
CMDPAR   DC    AL2(CMDPARE-CMDPAR)
         DC    CL2'  '
         DC    C'PAR GL,L='
CMDPARL  DC    CL54' '
CMDPARE  EQU   *
*
         DS    0H
CMDSHOW  DC    AL2(CMDSHOWE-CMDSHOW)
         DC    CL2'  '
         DC    C'SETF (0);SHOW L='
CMDSHOWL DC    CL54' '
         DC    C' TO 1'
CMDSHOWE EQU   *
*
SAVR14   DS    F
SAVR13   DS    F
SAVEAREA DS    18F
*
BLANKLIB DC    CL54' '
CNT0     DS    H                   COUNTER FOR OPEN ELEMENTS
CNT      EQU   CNT0+1
ZERO     DC    XL8'0000000000000000'
NOERR    EQU   ZERO
NEXT     DC    F'1'
*--  IEDTUPCB, IEDTAMCB, IEDTPARL ------------------------------------*
         IEDTUPCB C,,VERSION=2
         IEDTAMCB C
         IEDTPARL C,,VERSION=2
*--  PARAMETER LISTS FOR EDT CALLS ---------------------------------*
*--  PARAMETERS FOR IEDTEXE
CMDPL    DS    A                   A(EDTGLCB)
ACMD     DS    A                   A(STATEMENT)
*--  PARAMETERS FOR IEDTGTM
GTMPL    DS    A                   A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(AGTMPOS)          RECORD INDEX (IN)
         DC    A(AGTMKEY)          RECORD INDEX (OUT)
         DC    A(AGTMREC)          A(RECORD)
*--  PARAMETERS FOR IEDTPTM
PTMPL    DS    A                   A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(APTMKEY)
*--  PARAMETERS FOR IEDTGET -  LOCAL FILE DESCRIPTION
```

```
PARPL    DS    A                   A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(0)                DUMMY
         DC    A(0)                DUMMY
         DC    A(EDTPARL)
*
AGTMPOS  DS    A
AGTMREC  DS    A
AGTMKEY  EQU   AGTMPOS
APTMKEY  EQU   AGTMPOS
*
         DROP  R11
         END
```

```
*********************************************************************
* MARKS: EXAMPLE OF USING RECORD MARK 13                            *
*********************************************************************
*  FUNCTION:   INSERTS RECORDS WITH MARKS 13+15, WHICH INDICATE
*              THE RECORD MARKS.
*              THESE RECORDS ARE SKIPPED DURING WRITING TO
*              A FILE.
*              IG @PAR PROTECTION=ON HAS BEEN SET, THESE
*              RECORDS CANNOT BE OVERWRITTEN.
*  1) CHECK EDT VERSION
*  2) SYNTAX CHECK
*  3) INITIALIZE CONTROL BLOCKS
*  4) IEDTGTM: GET MARKED RECORDS
*  5) CHECK LAST DIGIT OF LINE NUMBER
*  6) MAKE MARK PRINTABLE
*  7) WRITE TEXT RECORD TO LINE NUMBER+0000.0001
*  8) DELETE LINE WITH MARK 13 FROM PREVIOUS CALL
*  9) RETURN TO EDT
*
*  ENTRY POINT: MARKS
*********************************************************************
MARKS    START
MARKS    AMODE ANY
MARKS    RMODE ANY
         GPARMOD 31
         SPACE
         EJECT
         TITLE 'MARKS'
*********************************************************************
*      DEFINITION SECTION
*********************************************************************
         REGS
EDTGLCB  IEDTGLCB  D
*
         EJECT
         ENTRY MARKS
*********************************************************************
*      STATEMENT SECTION
*********************************************************************
* INTERFACES: (SEE CALLS FOR EXTERNAL COMMANDS)
*   1ST PARAMETER: A(EDTGLCB)
*   2ND PARAMETER: A(COMMANDSTRING)
*********************************************************************
MARKS    CSECT
         USING MARKS,R15
         B     12(R15)
NAME     DC    CL8'MARKS   '
         STM   R14,R12,12(R13)     SAVE REGISTERS
         ST    R13,SAVR13
         LA    R13,SAVEAREA        SAVE AREA FOR EDT CALLS
         DROP  R15
*
         LR    R11,R15             LOAD BASE
         USING MARKS,R11
         L     R10,0(R1)           1ST PARAMETER
         L     R9,4(R1)            2ND PARAMETER
         USING EDTGLCB,R10
         ST    R10,INFPL           READ A(EDTGLCB) FROM ENTRY
**
*  1) CHECK EDT VERSION ***********************************************
         LA    R1,INFPL
         L     R15,=V(IEDTINF)
         BALR  R14,R15
         CLC   EGLRMSGF(10),EDTV164
         BE    SYNTAX
         LA    R1,MLD0             INVALID EDT VERSION
         B     ERR
         EJECT
*  2) SYNTAX CHECK ****************************************************
SYNTAX   LA    R1,4(R9)            R1: BEGINNING OF TEXT
         LH    R2,0(R9)            R2: LENGTH OF TEXT
         SH    R2,=H'4'            DEDUCT HEADER LENGTH
         SPACE ,
* TAKE INTO ACCOUNT KEYWORD 'MARKS'
         LTR   R2,R2               CHECK PARAMETER LENGTH
         BZ    INIT                NO PARAMETERS ->
         BAL   R14,NXTCHAR         SKIP BLANKS
         B     INIT
         CH    R2,HKEYL
         BL    ERRSY
         CLC   0(5,R1),KEYWORD     KEYWORD 'MARKS'?
         BNE   ERRSY               NO, ERROR
         AH    R1,HKEYL
         SH    R2,HKEYL            SKIP KEYWORD
         LTR   R2,R2
         BZ    INIT
         B     ERRSY               SYNTAX ERROR ->
         SPACE 2
* AUXILIARY ROUTINE
NXTCHAR  CLI   0(R1),C' '          BLANKS?
         BNE   4(0,R14)            NO ->
NXTCHAR1 LA    R1,1(0,R1)          (R1)=(R1)+1
         BCT   R2,NXTCHAR          (R2)=(R2)-1
         BR    R14                 END OF INPUT ->
         SPACE 2
         EJECT
```

```
*  3) INITIALIZE CONTROL BLOCKS ***************************************
INIT     ST    R10,PUTPL           READ A(EDTGLCB) FROM ENTRY
         ST    R10,GTMPL
         ST    R10,DELPL
         MVI   EAMMODB,EAMMOVM     MOVE MODE
         MVC   EAMPREC,=H'256'     LENGTH OF OUTPUT BUFFER
         XC    EAMDISP,EAMDISP
         LH    R2,=H'8'            SUPPLY KEY LENGTH
         STH   R2,EAMLKEY1         INPUT
         STH   R2,EAMLKEY2         INPUT
         STH   R2,EAMPKEY          OUTPUT
         MVC   EAMLKEY(4),EAMPKEY
         MVC   EAMFILE,EGLFILE     INSERT CURRENT WORK FILE
         XC    EAMKEY(8),EAMKEY    READ FIRST RECORD
*
         MVC   COUNTER,P0          INITIALIZE COUNTER
         MVC   NUMBER(8),=CL8' 0       '
         SPACE ,
*  4) IEDTGTM: GET MARKED RECORDS *************************************
GETLINE  EQU   *
         LA    R1,GTMPL
         L     R15,=V(IEDTGTM)
         BALR  R14,R15
         CLC   EGLMRET,NOERR
         BNE   ERREAM
*
         CLI   EGLSR1,EAMOK12      EOF REACHED?
         BNE   GETLINO             NO ->
         CLI   EAMDISP+3,1         AFTER TEXT HAS BEEN WRITTEN?
         BE    ENDE                YES, FINISHED
         SPACE ,
*  5) CHECK LAST DIGIT OF LINE NUMBER *********************************
GETLINO  CLI   EAMKEY+7,C'0'       CHECK IF LINE CAN BE INSERTED
         BE    KEYOK
         TM    EAMMARK2,EAMMK13    RECORD WITH MARK 13?
         BZ    ERR1                NO, FUNCTION NOT POSSIBLE
         B     DELTEXT             YES, CAN BE DELETED
         SPACE ,
KEYOK    MVC   ZLNR(4),EAMKEY      ENTER LINE NUMBER IN TEXT
         MVC   ZLNR+5(4),EAMKEY+4
         LA    R4,TEXTMARK
         MVI   0(R4),C' '
         MVC   1(39,R4),0(R4)
*  6) MAKE MARK PRINTABLE *********************************************
         LH    R5,EAMMARK
         LA    R3,16
LOOPMARK STC   R5,BYTE
         TM    BYTE,X'01'          BIT SET?
         BZ    NXTMARK             NO, NEXT
         LR    R2,R3
         BCTR  R2,0                (R2) = (R2)-1
         SLL   R2,1                (R2) = (R2)*2
         LA    R2,MARKTAB(R2)
         MVC   0(2,R4),0(R2)       GET DIGITS FROM TABLE
         LA    R4,2(R4)
         MVI   0(R4),C','          ENTER COMMA
         LA    R4,1(R4)
NXTMARK  SRL   R5,1                SHIFT TO NEXT BIT
         BCT   R3,LOOPMARK
         BCTR  R4,0
         MVI   0(R4),C' '          LAST COMMA SUPERFLUOUS
*  7) WRITE TEXT RECORD TO LINE NUMBER+0000.0001 *********************
         PACK  LINENUMB(8),EAMKEY(8) CONVERT KEY TO DECIMAL NUMBER,
         AP    LINENUMB(8),INCR(8)   INCREMENT BY 0.0001
         UNPK  EAMKEY(8),LINENUMB(8)
         OI    EAMKEY+7,X'F0'        TAKE SIGN INTO ACCOUNT
         MVI   EAMMARK1,0            SET WRITE PROTECTION PLUS
         MVI   EAMMARK2,EAMMK13+EAMMK15     IGNORE INDICATOR
         MVC   EAMLREC,=H'80'        RECORD LENGTH
         LA    R1,PUTPL
         L     R15,=V(IEDTPUT)
         BALR  R14,R15
         CLC   EGLMRET,NOERR
         BNE   ERREAM
         MVI   EAMDISP+3,1         START SEARCH AT NEXT RECORD
         AP    COUNTER,INCR        INCREMENT COUNTER
         B     GETLINE             READ NEXT MARKED LINE
         SPACE ,
*  8) DELETE LINE WITH MARK 13 FROM PREVIOUS CALL ********************
DELTEXT  LA    R1,DELPL
         L     R15,=V(IEDTDEL)
         BALR  R14,R15
         MVI   EAMDISP+3,1         START SEARCH AT NEXT RECORD
         B     GETLINE             READ NEXT MARKED LINE
         EJECT
* ERROR HANDLING + MESSAGE PROCESSING
ERREAM   CLI   EGLSR1,EAMAC16      FILE EMPTY?
         BE    RETURN              YES ->
         CLI   EGLSR1,EAMAC20      NO MARKS?
         BE    RETURN              YES ->
         CLI   EGLSR1,EAMAC28      FILE OPENED REAL?
         BE    RETURN              YES ->
         LH    R9,EGLRMSGL
         LTR   R9,R9               EDT MESSAGE?
         BZ    ERR1
         SH    R9,=H'10'           REMOVE '%  EDTNNNN'
         STH   R9,EGLRMSGL         CORRECT LENGTH
         BCTR  R9,0
         MVC   EGLRMSGF(L'EGLRMSGF-10),EGLRMSGF+10 MOVE MESSAGE
ERRSY    MVC   EGLMRET,=AL2(EUPSYERR)  BACK WITH ERROR
         B     RETURN2
         SPACE 2
ERR0     LA    R1,MLD0             "INVALID EDT VERSION"
         B     ERR
ERR1     LA    R1,MLD1             "FUNCTION ABORTED"
         B     ERR
         SPACE ,
ERR      MVI   EGLSR1,0
         MVC   EGLMRET,=AL2(EUPSYERR)
         B     RETURN1
         SPACE 2
ENDE     EQU   *
         UNPK  NUMBER(8),COUNTER(8)  MAKE COUNTER PRINTABLE
         OI    NUMBER+7,X'F0'
RETURN   MVI   EGLSR1,0            NO ERROR
         MVC   EGLMRET,=AL2(EUPRETOK)
         LA    R1,MSGOK
RETURN1  XR    R2,R2
         ICM   R2,3,0(R1)
         SH    R2,=H'4'            DEDUCT HEADER
         STH   R2,EGLRMSGL         ENTER LENGTH FIELD
         MVC   EGLRMSGF,4(R1)      TRANSFER MESSAGE TEXT
*  9) RETURN TO EDT ***************************************************
RETURN2  L     R13,SAVR13
         LM    R14,R12,12(R13)
         BR    R14                RETURN TO EDT
         SPACE
         EJECT
         LTORG
         SPACE 2
SAVEAREA DS    18F                SAVE AREA
SAVR13   DS    F
HKEYL    DC    H'5'                LENGTH OF ENTRY NAME
NOERR    DC    H'0'
KEYWORD  DC    C'MARKS'
*
MLD0     DC    AL2(MLD0E-MLD0)
         DC    CL2'  '
         DC    C'ROUTINE NOT EXECUTABLE BELOW EDT V16.4A'
MLD0E    EQU   *
MLD1     DC    AL2(MLD1E-MLD1)
         DC    CL2'  '
         DC    C'FUNCTION ABORTED'
MLD1E    EQU   *
```

```
MSGOK    DC    AL2(MSGOKE-MSGOK)
         DC    CL2'  '
         DC    C'NUMBER OF MARKED RECORDS:        '
MSGOKE   EQU   *
NUMBER   EQU   MSGOK+33
*
         DS    0F
TEXT     DC    C'*** LINE NUMBER '
ZLNR     DC    CL9'0000.0000'
         DC    C':  MARK '
TEXTMARK DS    CL40
** PARAMETER BLOCK FOR IEDTINF CALL
INFPL    DS    A                  A(EDTGLCB)
** PARAMETER BLOCK FOR IEDTPUT CALL
PUTPL    DS    A                  A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(EAMKEY)
         DC    A(TEXT)
** PARAMETER BLOCK FOR IEDTGTM CALL
GTMPL    DS    A                  A(EDTGLCB)
         DC    A(EDTAMCB)
         DC    A(EAMKEY)            INPUT
         DC    A(EAMKEY)            OUTPUT
         DC    A(EDTREC)
** PARAMETER BLOCK FOR IEDTDEL CALL
DELPL    EQU   GTMPL              A(EDTGLCB)
*
LINENUMB DC    PL8'0'
INCR     DC    PL8'1'              CORRESPONDS TO INC=0000.0001
P0       DC    PL8'0'
COUNTER  DS    PL8                 NUMBER OF MARKED RECORDS
EAMKEY   DS    D
EDTREC   DS    CL256
MARKTAB  DC    C'15141312111009080706050403020100'
BYTE     DS    X
EDTV164  DC    C'EDT V16.4A'
         EJECT
         PRINT GEN
         IEDTAMCB C
         DS    0F
         DROP  R11,R15
         EJECT
         SPACE
         ETPND KLGR,VER=164,DATE=&SYSDATE,PATCH=256
         END
```

5 Calling a user program: @RUN

@RUN can be used to load a user program which is available as a module (see the description of @RUN in the "EDT Statements" manual [1]).

Loading the user program as a subroutine

The module to be loaded is loaded into any free area in the virtual address space by the Dynamic Linking Loader (DLL). External references and V-type constants are resolved from the specified library in accordance with the DLL function.

If UNLOAD is not specified, the module remains loaded.

This means that the module does not need to be loaded again when a further call for it is issued.

If a module is called several times without UNLOAD, care should be taken when using variables that they are initialized explicitly.

If there are several modules with the same CSECTs or entry points (ENTRY names), the last of these is loaded.

Unloading the user program

The module can be unloaded by means of @RUN (operand UNLOAD) or @UNLOAD.

If the name of a control section (CSECT) or of an entry point (ENTRY) which differs from the name of the associated module is specified, the module cannot be unloaded by means of UNLOAD. In this case, the module must be unloaded in the called program itself either using the UNBIND macro or by means of @UNLOAD.

The same applies to modules which are loaded dynamically in the called program either explicitly via the BIND macro or using the autolink function of DLL.

i If a user program that can only run in 24-bit addressing mode is to be called, EDT must be called as follows:

```
START-PROGRAM *MOD($EDTLIB,EDTC)
```

If the user program is available as a load unit of BLS, the START-PROGRAM statement must contain the operand RUN-MODE=ADVANCED.

Information passed to the user program

After the module has been loaded, it is started at the specified entry point. EDT passes the contents of the following registers to the program:

Register	Contents
1	The address of the first line in the current virtual file, or 0 if the file is empty.
2	The address of the first byte of the parameter (string) being passed, or 0 if nothing is passed.
3	(length-1) of the parameter being passed. If, however, register 2 contains 0, the contents of register 3 are meaningless.
4	The address of entry point ENTRLINE.
5	The address of entry point FINDLINE.
6	The address of entry point DELETE.
7	The address of the FILELINE buffer in EDT.
13	The address of an 18-word save area.
14	The EDT return address.
15	The start address in the called program.

All registers should be saved when the subroutine is called and should be restored again before control is returned to EDT via register 14.

Processing the current work file

When processing the current work file, EDT passes the address of the first line in this file in register 1 (provided the file is not empty).

A line in the virtual work file has the following internal structure:

```
FILELINE    DESECT                  Description of a virtual EDT line:
LINENUMB    DS          XL4         Line number
BACKLINK    DS          XL4         Address of the previous line or zero
FWDLINK     DS          XL4         Address of the next line or zero
            DS          XL3         Reserved
LINELGN     DS          XL1         Length of the line contents minus 1
IMAGE       DS          256C        The contents of the line
```

The first line in a file has a BACKLINK of zero, the last line a FWDLINK of zero.

The address of FILELINE is not necessarily a word address. For this reason, BACKLINK and FWDLINK may contain addresses of FILELINE buffers which are not aligned on a word boundary.

In order to permit processing of these lines, EDT provides three routines to which the subroutine may branch:

1. the ENTRLINE routine for inserting lines
2. the FINDLINE routine for finding lines
3. the DELETE routine for deleting lines.

For these routines - in contrast to normal programming practices - certain register conventions must be observed. These are explained below for each of the routines.

1. ENTRLINE routine   Insert lines

The FILELINE buffer describes precisely one virtual line. The line to be inserted must be passed to EDT in this FILELINE buffer.

The address of the buffer is in register 7 (USING FILELINE,7).

The following fields in the FILELINE buffer must be supplied with data:

– LINENUMB must contain the decimal packed line number. The line number 34.05, for example, must be stored as X'00340500' in LINENUMB.
– LINELGN must contain the length of the line being passed.
– IMAGE must contain the contents of the new line.

EDT places the entry point for the ENTRLINE routine in register 4.
The return address must be placed in register 9.

ENTRLINE call: BALR 9,4

Control returns from EDT to the subroutine after 0(0, 9) in the case of successful execution, otherwise after 4(0, 9).

Registers 8, 10 and 13 are modified by EDT; all other registers remain unchanged. After execution, the address of the inserted line is in register 8.

2. FINDLINE routine   Find lines

This routine searches for a line whose line number is equal to or greater than a specified line number. If there are several such lines, the one with the lowest line number is chosen. The number of the desired line must be stored in decimal packed format, without a sign, in register 2.

EDT places the entry point address of the FINDLINE routine in register 5.

The return address must be placed in register 14.

FINDLINE call: BALR 14,5

The return from EDT to the subroutine is after 0(0,14) if the search was successful, otherwise after 4(0,14).

All registers except register 7 remain unchanged. If the search was successful, register 7 contains the address of the line whose line number is equal to or greater than the number specified in register 2.
If no line is found, the contents of register 7 also remain unchanged.

3. DELETE routine Delete lines

The following registers must be set:

– register 2 contains the line number of the first line to be deleted,

– register 3 contains the line number of the last line to be deleted.

Both line numbers must be in decimal packed format, without signs.

The line number in register 3 must not be less than that in register 2.

EDT places the entry point address of the DELETE routine in register 6.

The return address must be placed in register 4.

DELETE call: BALR 4,6

Control returns from EDT to the subroutine after 0(0, 4) in the case of successful execution, otherwise after 4(0, 4).

All registers remain unchanged.

If errors occur during execution of these routines, control is passed to EDT (e.g. after an REQM ERROR when inserting a line).
Further results are unpredictable.

Example 1

The program is to attempt to generate line 56.4321 and to write the TSN in this line, regardless of whether or not there are already lines in the virtual file.

```
XMPL1     CSECT
XMPL1     AMODE ANY
XMPL1     RMODE ANY
          GPARMOD 31
          STM    14,12,12(13)
          BALR   10,0
          USING  *,10
          USING  FILELINE,7 -------------------------------------------- (01)
          ST     13,REG13
          TMODE  PARLIST=TSKBER
          L      1,REG1
          MVC    IMAGE(23),TEXT ---------------------------------------- (02)
          MVC    IMAGE+23(4),TSKTSN ------------------------------------ (03)
          MVI    LINELGN,X'1A' ----------------------------------------- (04)
          MVC    LINENUMB,=X'00564321' --------------------------------- (05)
          BALR   9,4 --------------------------------------------------- (06)
          L      13,REG13
          LM     14,12,12(13)  ----------------------------------------- (07)
          B      0(0,14) ----------------------------------------------- (08)
TSKBER    DS     0A
          DTMODE DSECT=NO
REG13     DS     F
TEXT      DC     'THIS TASK HAS THE TSN '
FILELINE  DSECT
LINENUMB  DS     XL4     LINE NUMBER
BACKLINK  DS     XL4     ADDRESS OF PREVIOUS LINE
FWDLINK   DS     XL4     ADDRESS OF NEXT LINE
          DS     XL3     RESERVED
LINELGN   DS     XL1     LENGTH OF LINE CONTENTS MINUS 1
IMAGE     DS     256C    CONTENTS OF THE LINE
          END
```

(01) The FILELINE buffer describes precisely one virtual line. The line to be inserted must be passed to EDT in this FILELINE buffer.
The address of the buffer is in register 7 (USING FILELINE,7).

(02) The text "THIS TASK HAS THE TSN " is placed in the IMAGE field of the FILELINE buffer.

(03) The text in the IMAGE field is extended to include the current TSN.

(04) The (length-1) of the line image is placed in the LINELGN field.

(05) The line number is placed in the LINENUMB field.

(06) EDT keeps the entry point address of routine ENTRLINE in register 4. The return address must be placed in register 9.

(07) After the return from this routine, the EDT register contents are restored.

(08) Finally, control is returned to EDT.

The subroutine XMPL1 is now to be called from EDT:

```
   23.00 .......................................................................
@run (xmpl1,lib1) ................................................0000.00:001(0)
LTG                                                       TAST
```

The subroutine XMPL1 in library LIB1 is executed, causing line 56.4321 to be created in the work file.

```
56.43 THIS TASK HAS THE TSN 1234...........................................
57.43 ..........................................................................
```

Example 2

Subroutine XMPL2 is to create a new line with the specified text in the EDT work file from which it is called.

```
XMPL2     CSECT
XMPL2     AMODE ANY
XMPL2     RMODE ANY
          GPARMOD 31
          STM    14,12,12(13)
          USING  *,10
          USING  FILELINE,7 -------------------------------------------- (01)
          ST     13,REG13
          MVC    LINELGN(1),3(3) --------------------------------------- (02)
          MVC    IMAGE(256),0(2) --------------------------------------- (03)
          MVC    LINENUMB,=X'07777700' --------------------------------- (04)
          BALR   9,4 --------------------------------------------------- (05)
          L      13,REG13
          LM     14,12,12(13)
          B      0(0,14) ----------------------------------------------- (06)
REG13     DS     F
FILELINE  DSECT
LINENUMB  DS     XL4    LINE NUMBER
BACKLINK  DS     XL4    ADDRESS OF PREVIOUS LINE
FWDLINK   DS     XL4    ADDRESS OF THE NEXT LINE
          DS     XL3    RESERVED
LINELGN   DS     XL1    (LENGTH-1) OF THE LINE CONTENTS
IMAGE     DS     256C   CONTENTS OF THE LINE
          END
```

(01) The FILELINE buffer describes precisely one virtual line. The line to be inserted must be passed to EDT in this FILELINE buffer.
The address of the buffer is in register 7 (USING FILELINE,7).

(02) Register 3 contains the (length-1) of the parameter being passed. Only the last byte of this is relevant; it is placed in the length field of the line buffer.

(03) The parameter address is passed in register 2. Since the actual length of the line to be created is already known, the maximum length of 256 bytes is simply copied into the line buffer.

(04) The line number is placed in the LINENUMB field.

(05) EDT keeps the entry point address of routine ENTRLINE in register 4. The return address must be placed in register 9.

(06) Control returns to EDT.

The subroutine XMPL2 is now to be called from EDT:

```
   23.00 .........................................................................
@run (xmpl2,lib1) ........................................0000.00:001(0)
LTG                                                         TAST
```

Subroutine XMPL2 in library LIB1 is called and the string ’ NEW LINE ’ is passed to it. The routine then creates line 777.77 in the work file.

```
777.77  NEW LINE...........................................................
778.77 ....................................................................
```

6 L mode subroutine interfaces

An application program can call EDT as a subroutine. In the BIND macro call SYMBOL=EDT must be specified as the entry point address and LIBNAM=$EDTLIB as the link module library.

When EDT is called:

– register 1 must contain the address of a two-word parameter list
– register 13 must contain the address of an 18-word save area
– register 14 must contain the return address
– register 15 must contain the entry point address of EDT.

The entry point address to EDT is passed by the BIND macro in the field specified by the SYMBLAD operand. The parameter list and the save area must be defined in the calling user program.

In the save area EDT saves the registers of the calling program. Before branching to EDT the calling program must ensure that the parameter list has those contents required for the desired EDT run.

If BS2000/OSD V2.0 or higher is available and EDT is installed with IMON, it is possible to explicitly call a particular EDT version from the application program.

Proceed as follows to do this:

– obtain the full version number using the IMON function GETPVER

– obtain the installation file name of the EDT module library (PRODNAM=EDT, LOGID= SYSLNK,PRODVER=<result GETPVER>) using the IMON function GETINSP

– load the EDT prelinked module with
BIND (SYMBOL=EDT,LIBNAME=<result GETINSP>)

For existing applications the highest available EDT version is linked.

A call for EDT in a user program could be programmed as follows:

```
          L     15,EDTADDR ---------------------------------------------- (01)
          LTR   15,15 --------------------------------------------------- (02)
          BNZ   CALLEDT ------------------------------------------------- (03)
          BIND  MF=E,PARAM=BINDPAR ------------------------------------- (04)
          USING BINDSECT,1
          CLC   XBINRET,=X‘00000000‘ ---------------------------------- (05)
          BNE   BINDERR
          L     15,EDTADDR ---------------------------------------------- (06)
CALLEDT   LA    13,SAVEAREA --------------------------------------------- (07)
          LA    1,PLIST
          BALR  14,15

BINDPAR   BIND  SYMBOL=EDT,LIBNAME=$EDTLIB,SYMTYP=CSEN,SDINFO=DEF, -
                SYMBLAD=EDTADDR,MF=L
BINDSECT  BIND  MF=D,REFIX=X
```

(01) EDTADDR initially contains zero. After EDT has been loaded, the EDT entry point address is placed here.

(02) Does register 15 contain a value other than zero, i.e. has the EDT entry point address been loaded?

(03) If register 15 contains the entry point address, the program branches to the EDT call.

(04) EDT is loaded.

(05) If loading of EDT was successful, BIND supplies the return code 0.

(06) The entry point address to EDT was stored by BIND in EDTADDR and is loaded from there into register 15.

(07) The address of the save area is loaded into register 13 and that of the parameter list in register 1. Then a branch is made to EDT.

Structure of the parameter list

The parameter list for calling EDT as a subroutine must be defined in the calling program.

The address of the parameter list is passed in register 1. The list has the following structure:

	Byte0	Byte1	Byte2	Byte3
Word 1	opt1	opt2	f	
Word 2	d		e	

opt1	Function byte 1 (see “Functions of the function bytes” on page 140ff)
opt2	Function byte 2 (see “Functions of the function bytes” on page 140ff)
f	Page number Start of virtual address space Specifies (in hexadecimal mode) as of which page of virtual address space the virtual file is to begin. If the specified page is already reserved, the first free page whose number is greater than f is used. EDT evaluates the parameter f only when the first branch is made to EDT. Normally f and d should have the same value.
d	Page number Start of the data and variable area Specifies in hexadecimal as of which page of virtual address space a four-page area is to begin, in which the data and variables used by EDT are to be held. Normally f and d should have the same value. When first called, EDT writes the start address of the data and variable area into the second word of the parameter list. d is overwritten in doing so and may then not be changed again. EDT does not check whether the save area passed using the parameter d is already reserved. For this reason the REQM macro should be used to request a four-page area. The page number to be passed to EDT as parameter d should be formed from the start address supplied.
e	Reserved area When called for the first time, EDT writes the address of the data and variable area into the second word of the parameter list.

This format of the parameter list is expected if the initialization bit 2^0 (opt1) and bit 2^5 (opt2) are set.

Functions of the function bytes

opt1 (function byte 1)

controls execution of EDT. The first time EDT is called, bit 2^0 must be set. After the first call, EDT sets bit 2^0 to zero. For all subsequent calls, bit 2^0 must be zero. For each branch to EDT, at least one of the bits 2^1 to 2^7 must be set.

2^0:	Must be set for the first call of EDT after loading.
2^1:	EDT retains control until the user enters @RETURN. After returning to the calling program, the current file is deleted.
2^2:	EDT retains control until the user enters @RETURN. After returning to the calling program, the current file is retained.
2^3:	EDT reads only a single line and then returns control to the calling program. Any text line which is entered is inserted in the current file.
2^4:	EDT reads only a single line and then returns control to the calling program. Any text line which is entered is not inserted in the current file.
2^5:	EDT accepts the address of a line from the calling program in register 0 and places this line in the current file.
2^6:	EDT searches the current file for a line and then returns control to the calling program.
2^7:	A line range is deleted from the current file and control is then returned to the calling program.

opt2 (function byte 2)

controls execution of EDT for abortion, for record-by-record transfer of a file, and for supplying a return code.

2^0:	No statement is accepted before @PRINT has been completed.
2^1:	EDT is not unloaded after @HALT or if it is terminated abnormally.
2^2:	EDT accepts a file record-by-record from the calling program.
2^3:	EDT passes a file record-by-record to the calling program.
2^4:	Reserved; must be zero.
2^5:	Auxiliary initialization bit. Definition of the parameter list.
2^6:	The EDT STXIT routine is not set up.
2^7:	Reserved; must be zero.

Function byte 1

Bit	Function	Bits which may also be set
2^0	Initialization bit The first time EDT is called, bit 2^0 must be set. If bit 2^0 is the only bit set in function byte 1, it is not possible to return from EDT to the calling program. In all subsequent EDT calls, bit 2^0 must be off. Return codes Reg. 15: X'10' in the rightmost byte if an error was detected in the first call for EDT. Note If bit 2^5 in opt2 is also set, page numbers greater than 255 may be specified for initialization of the virtual address space (parameters d and f).	opt1: 2^1, 2^2, 2^3, 2^4, 2^5, 2^6,2^7 opt2: 2^0, 2^1, 2^2, 2^5, 2^6
2^1	After EDT has been called, control is returned to the calling program only when a @RETURN statement is entered at the main level. When this is done, the current virtual file or the file opened by means of @OPEN is deleted. Return codes opt2, 2^1 not set Reg.15: no return code provided (X'10' may occur after an initialization error if bit 20 in opt1 is set). Reg.1: X'00000000' Return using @RETURN, the file being processed has been deleted. opt2, 2^1 set Reg.15: X'00' in the rightmost byte Reg.1: X'00000000' Return using @RETURN, the file being processed has been deleted (see note).. Reg.15: X'04' Return using @HALT, the file being processed has not been deleted. Reg.1: Contents undefined. Reg.15: X'08' EDT ABNORMAL END has occurred. Reg.1: Contents undefined. Reg.15: X'0C' Unknown end condition, return code only if the EDT STXIT routine is called; otherwise end with dump. Reg.1: Contents undefined. Note If @HALT is entered and there are files in the virtual memory which have not been saved (main or work file), the user is asked whether he wants to save these files. If task switch 4 is set, EDT suppresses this query	opt1: 2^0, 2^5 opt2: 2^0, 2^1, 2^5, 2^6

Bit	Function	Bits which may also be set
	Caution: EDT does not release the memory space occupied by the virtual file. The virtual EDT file is still available for further processing. Exceptions: if a file was opened by means of @OPEN before @HALT is entered, this file is closed. A subsequent EDT call without the initialization bit set will then be rejected with EDT ABNORMAL END. If @HALT is encountered in an EDT (@DO) procedure or an INPUT file, a subsequent EDT call without the initialization bit set will likewise be rejected with EDT ABNORMAL END.	
2^2	After EDT has been called, control is returned to the calling program only when a @RETURN statement is entered at the main level. When this is done, the current virtual file or the file opened by means of @OPEN remains in memory. If the current file at the time of the return is in the virtual address space, the start address of the first line of this file is placed in register 1. If the current file is the disk file opened via @OPEN, the two's complement of the EDT FCB address, i.e. the negative FCB address, is placed in register 1. The calling program should not use this FCB. If it does, and a DMS error occurs as a result of this, the error handling routines assume that the registers contain the values they had when EDT was running. In actual fact, however, the registers contain other values. This will lead to unpredictable results. If the current file at the time of return is empty, the value zero is placed in register 1. Return codes opt2, 2^1 not set Reg.15: no return code provided (X'10' may occur after an initialization error if bit 2^0 in opt1 is set). Reg.1: X'00000000' (current file is empty) or the start address if the current file is in the virtual address space. opt2, 2^1 set Reg.15: X'00'. Reg.1: X'00000000' or address Return using @RETURN. Reg.15: X'04' Return using @HALT (see note for opt1, 2^1). Reg.1: Contents undefined. Reg.15: X'08' EDT ABNORMAL END. Reg.1: Contents undefined. Reg.15: X'0C' Unknown end condition, return code only if the EDT STXIT routine is called; otherwise end with dump. Reg.1: Contents undefined.	opt1: 2^0, 2^5 opt2: 2^0, 2^1, 2^5, 2^6

Bit	Function	Bits which may also be set
2^3	After EDT has received control, the user can enter only a single line, and control is then returned to the calling program. If a text line is entered, this line is inserted in the current file (the virtual file or the file opened via @OPEN). When control is returned to the calling program, register 1 contains the start address of the line which was inserted. If an EDT statement is entered, EDT executes this statement and returns control to the calling program. In this case, register 1 contains 0 and register 0 contains the address of a byte which contains the current EDT statement symbol (@ or the symbol defined by @:edtsymb). Return codes opt2, 2^1 not set Reg.15: no return code provided (X'10' may occur after an initialization error if bit 2^0 in opt1 is set). Reg.1: X'00000000' or an address. Reg.0: an address (see above). opt2, 2^1 set Reg.15: X'00' in the rightmost byte. Reg.1: see above. Reg.0: see above. Return after processing. Reg.15: X'04' Return using @HALT (see note for opt1, 2^1). Reg.1: Contents undefined. Reg.0: Contents undefined. Reg.15: X'08' EDT ABNORMAL END. Reg.1: Contents undefined. Reg.0: Contents undefined. Reg.15: X'0C' Unknown end condition, return code only if the EDT STXIT routine is called; otherwise end with dump. Reg.1: Contents undefined. Reg.0: Contents undefined. Reg.15: X'10' Initialization error, can occur only if opt1, 2^0 is set. Reg.1: Contents undefined. Reg.0: Contents undefined. Notes If a statement was processed, register 1 contains the value 0 and register 0 points to a byte which contains the current EDT statement symbol.	opt1: 2^0, 2^5 opt2: 2^0, 2^1, 2^5, 2^6

Bit	Function	Bits which may also be set
	If @RETURN was processed, the contents of register 0 are undefined and the contents of register 1 depend on the file being processed. If there is a file in the virtual address space when control is returned, register 1 contains the start address of the first record in this file. If the current file was opened via @OPEN, the two's complement of the EDT FCB address is placed in register 1. If the current file is empty, register 1 contains 0.	
2^4	After EDT has received control, the user can enter only a single line, and control is then returned to the calling program. If a text line is entered, EDT does not insert this line in a file. However, EDT increments the current line number and places the start address of this dummy line (which would have resulted from inserting a line) in register 1 before control is returned. If an EDT statement is entered, EDT executes this statement and then returns control to the calling program. When this is done, register 1 contains the value 0 and register 0 contains the address of a byte which contains the current EDT statement symbol (@ or the symbol defined by @:edtsymb). Return codes: see above (opt1, 2^3)	opt1: 2^0, 2^5 opt2: 2^0, 2^1, 2^5, 2^6
2^5	After EDT has received control, it reads in the line whose start address the calling program has passed in register 0. The line must be a variable-length record, i.e. the first halfword contains the length, followed by two blanks and then the actual record. The record begins at a halfword boundary. The first four bytes must be included in the record length, and the record length should be at least 5 since EDT will otherwise ignore the record, place 0 in register 1 and return to the calling program. EDT processes the record like a normal input (text line or EDT statement). If, in addition to bit 2^5, bit 2^3 or 2^4 is set, EDT processes the record in accordance with the conventions for this bit and then returns control to the calling program. When this is done, the contents of registers 0 and 1 comply with the conventions for bit 2^3 or 2^4, as applicable. If, in addition to bit 2^5, bit 2^1 or 2^2 is set, EDT processes the record passed by the calling program and then reads further inputs from SYSDTA or from the terminal. As soon as the @RETURN statement is entered, control is returned to the calling program in accordance with the conventions for bit 2^1 or 2^2, as applicable. If, in addition to bit 2^5, none of the bits 2^1, 2^2, 2^3 or 2^4 is set, then this has the same effect as setting bit 2^2 in addition to bit 2^5. Return codes see above (opt1, 2^3 or opt1, 2^2)	opt1: 2^0, 2^1, 2^2, 2^3, 2^4 opt2: 2^0, 2^1, 2^5, 2^6
	Notes EDT statement EDT accepts lowercase letters in a statement only if @LOWER ON has been entered. If the statement passed to EDT results in an error, no information is provided at the subroutine interface.	

Bit	Function	Bits which may also be set
	Record Any tab characters in the record are evaluated. The record is processed in accordance with the input mode (@INPUT CHAR/HEX/BINARY). Blocked input (records separated by X’15’; block mode ON) is not possible. If @LOWER OFF is set, lowercase letters are not converted to uppercase. If the record causes an error (e.g. MAX LINENUMBER, REQM ERROR, SOME INPUT TRUNCATED, etc.), no information is provided at the subroutine interface.	
2^6	After EDT has received control, it searches the current file for the line whose number was passed by the calling program. When EDT is called, register 0 must contain the address of a word (starting at a word boundary) which contains the desired line number in the form of a packed, unsigned decimal number. Line number 123.041, for example, must be stored in the word as X’01230410’. When EDT has completed the search, it returns control to the calling program. If EDT has found no line with a number equal to or greater than the specified line number, register 15 contains 4 when control is returned. If EDT has found a matching line, register 15 contains 0 and register 1 contains the address of the line whose number is equal to or greater than the specified line number. If the file contains several lines which satisfy the condition, register 1 contains the address of the line with the lowest number equal to or greater than the specified line number. Return codes opt2, 2^1 not set Reg.15: X’00’ Record found. Reg.1: see above. Reg.0: see above. Reg.15: X’04’ Record not found. Reg.1: Contents undefined. Reg.0: see above. Reg.15: X’10’ Initialization error, can occur only if opt1, 2^0 is set. opt2, 2^1 set Reg.15: X’00’ Record found. Reg.1: see above. Reg.15: X’04’ Record not found. Reg.1: Contents undefined. Reg.15: X’08’ EDT ABNORMAL END. Reg.1: Contents undefined.	opt1: 2^0 opt2: 2^0, 2^1, 2^5, 2^6

Bit	Function	Bits which may also be set
	Reg.15: X’0C’ Unknown end condition, return code only if the EDT STXIT routine is called; otherwise end with dump. Reg.1: Contents undefined. Reg.15: X’10’ Initialization error, can occur only if opt1, 2^0 is set.	
2^7	After EDT has received control, it deletes a line range in the current file (virtual file or the file opened using @OPEN). The calling program passes the address of a doubleword, aligned on a word boundary, in register 0. The first word contains the number of the first line to be deleted, in the form of an unsigned packed decimal number. The second word contains, in the same form, the number of the last line to be deleted. The number in the second word must not be less than the number in the first word. After EDT has deleted the line range, it returns control to the calling program with the result of the delete operation in register 15. If register 15 contains 4, the entire file has been deleted. If register 15 contains 0, the entire line range has been deleted, but there are still lines in the file. Return codes opt2, 2^1 not set Reg.15: X’00’ The line range has been deleted, there are still lines in the file. Reg.15: X’04’ The line range has been deleted and the file is now empty. Reg.15: X’10’ Initialization error, can occur only if opt1, 2^0 is set. opt2, 2^1 set Reg.15: X’00’ The line range has been deleted, there are still lines in the file. Reg.15: X’04’ The line range has been deleted and the file is now empty. Reg.15: X’08’ EDT ABNORMAL END. Reg.15: X’0C’ Unknown end condition; return code only if the EDT STXIT routine is called, otherwise end with dump. Reg.15: X’10’ Initialization error; may occur only if opt1, 2^0 is set.	opt1: 2^0 opt2: 2^0, 2^1, 2^5, 2^6

Function byte 2

Bit	Function	Bits which may also be set
2^0	If bit 2^0 is not set, EDT will accept a statement entered during section-by-section output using @PRINT... V. If bit 2^0 is set, EDT will reject a statement entered when the prompt *+-0 is displayed.	opt1: 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, opt2: 2^0, 2^2, 2^3, 2^5, 2^6
2^1	If this bit is set, EDT is not unloaded after a @HALT statement or after abnormal termination. Whenever control is returned to the calling program, a return code is passed in the rightmost byte of register 15. Return codes (see table below) X’08’ EDT ABNORMAL END X’0C’ Unknown end condition. Return code only if EDT STXIT routine is called, otherwise end with dump. X’10’ Initialization error, can occur only if opt1, 2^0 is set. Note Abortion by means of K2, @SYSTEM, @LOAD, @EXEC, @RUN (without return to EDT) cannot be prevented by this.	opt1: 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7 opt2: 2^0, 2^1, 2^2, 2^3, 2^5, 2^6
2^2	EDT is called to accept a file from the calling program. The file is passed record by record. The address of the record being passed is in register 0. The records are variable-length records. In EDT, each record receives the current line number, which is then incremented by the current increment.	opt1: 2^0 opt2: 2^1, 2^5, 2^6

Bits also set	X’00’	X’04’
opt1, 2^1	Return using @RETURN File deleted	Return using @HALT File not deleted
opt1, 2^2	Return using @RETURN (see description for opt.1, 2^2)	Return using @HALT
opt1, 2^6	Record found	Record not found
opt1, 2^7	Line range deleted, still lines in file	Lane range deleted, file is empty

Bit	Function	Bits which may also be set
	Return codes Reg. 15: X’00’ Record transferred to the file. Reg. 15: X’04’ Error occurred during processing of the record (e.g. MAXLINE NUMBER) Reg. 15: X’08’ EDT ABNORMAL END (only if opt2, 2^1 is set). Reg. 15: X’10’ Initialization error in first EDT call (only if opt1, 2^0 is set). Record format The record is a variable-length record and must begin on a halfword boundary. The calling program must place the address of the record in register 0. Byte 1-2: Record length (including record length field) Byte 3-4: Reserved (zero or blanks) Byte 5-260: Text Notes This function can be used to write records from the calling program into the EDT work file or procedure files. The work file may be a virtual file or a file opened by means of @OPEN. If there is already a line with the line number generated for the new record, then this line is overwritten. If the file already contains lines, the line number can also be generated in sequential mode (@EDIT SEQUENTIAL). In this case, existing lines are not overwritten. Records which begin with the current statement symbol are not regarded as statements, but are simply placed in the file (in contrast to bit 2^5 of opt1). The records being read are not examined for tab characters. The record being read is interpreted, depending on the input mode, as a string of printable characters (@INPUT CHAR; default value), a string of hexadecimal characters (@INPUT HEX), or a string of binary characters (@INPUT BINARY). Records with zero characters or more than 256 characters are not accepted (return code X’04’). The EDT STXIT routine is not set up for this call.	
2^3	EDT is called to pass a file record-by-record to the calling program. Register 0 contains the address of the record. The record is a variable-length record. The first call in which this bit is set transfers the record with the current line number. If there is no record for the current line number, the record with the next higher line number is transferred. In each subsequent call, the next record from the file is transferred. When the end of the file is reached, register 0 is set to 0.	opt1: 2^0 opt2: 2^1, 2^5, 2^6

Bit	Function	Bits which may also be set
	Return code Reg.15: in the rightmost byte X’00’ Record transferred (address in register 0) or end of file was reached (register 0 = 0). X’04’ No valid record found (e.g. FILE IS EMPTY, or current line number > $). X’08’ EDT ABNORMAL END (only if opt2, 2^1 is set). X’10’ Initialization error in first EDT call if opt1, 2^0 is set. Record format The record is a variable-length record and does not begin on a halfword boundary. The address of the record is passed in register 0. Byte 1-2: Record length (including record length field) Byte 3-4: Zeros Byte 5-260: Text If register 0 contains 0, the end of the file has been reached. Notes If the entire file is to be transferred, the current line number must be set to the line number of the first record in the file before the first EDT call with this bit set is issued (@SET%). This can be done from the calling program via opt1, 2^4 and 2^5. Record transfer can be interrupted before the end of the file is reached. If the function is called again, transfer resumes at the current line number, which is not modified by this call. If the current line number (*) is less than the line number of the first record in the file (%) when this call is issued, the first record is transferred. If * is greater than the line number of the last record in the file ($), EDT returns control with return code X’04’. The EDT STXIT routine is not set up for this call.	
2^4	Reserved; must always be zero.	
2^5	Auxiliary initialization bit. If this bit is set, the address of the virtual file and the address of the data area are passed in two halfwords of the parameter list. The format of the parameter list is: (see below)	opt1: 2^0, 2^1, 2^2, 2^3, 2^4, 2^5 opt2: 2^0, 2^1, 2^2, 2^6

	Byte 0	Byte 1	Byte 2	Byte 3
Word 1	opt1	opt2	f	
Word 2	d		e (reserved)	

Bit	Function	Bits which may also be set
	If this bit is not set, EDT expects the parameter list in the format: | | Byte 0 | Byte 1 | Byte 2 | Byte 3 | Word 1: opt1 | opt2 | reserved | f Word 2: d | reserved Note This bit is evaluated only during initialization of EDT. It must be set together with opt1, 2^0.	
2^6	If this bit is additionally set in an EDT call, the EDT STXIT routine is not set up. If this bit is not set, EDT sets up its STXIT routine again in each call. The routine is deactivated before control is returned to the calling program. Notes If this bit is set together with opt1, 2^1, 2^2, 2^3, 2^4 or 2^5, it should be noted that EDT procedures or @INPUT files can be aborted by means of K2 or SEND-MESSAGE only if the EDT STXIT routine is set up. If specified with opt2, 2^2 or 2^3, this bit has no effect since the EDT STXIT routine is never set up for these calls.	opt1: 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7 opt2: 2^0
2^7	Reserved; must always be zero.	

STXIT routine

EDT includes a STXIT routine which is set up anew each time EDT is called (exceptions: see opt2, 2^2, 2^3, 2^6). If the user program has its own STXIT routine, then this must also be set up again each time control is returned from EDT. If this is not done, the EDT STXIT routine would remain effective after the return from EDT, e.g. for messages to the program, program errors and unrecoverable error interrupts, and could lead to unpredictable results. It is also advisable to deactivate the user STXIT routine for console interrupts by means of CLOSE before calling EDT for the first time, so that initialization of EDT can be completed without interruption.

No EDT STXIT routine for ESCPBRK is defined at this interface.

Example for STXIT at LU15 interface

```
PROG1    CSECT
PROG1    AMODE ANY
PROG1    RMODE ANY
         GPARMOD 31
R0       EQU   0
R1       EQU   1
R10      EQU   10
R13      EQU   13
R14      EQU   14
R15      EQU   15
         PRINT NOGEN
* STATEMENT SECTION
         BALR  R10,R0
         USING *,R10
LOOP     WRTRD WRITE,,READ,,6,TERM ------------------------------------ (01)
         CLC   MESSG,=‘ST‘
         BE    TERM
         PACK  MESSG(3),MESSG(3) --------------------------------------- (02)
         MVC   OPT1,MESSG+1
         L     R15,EDTADDR
         LTR   R15,R15
         BNZ   CALLEDT
         BIND  MF=E,PARAM=BINDPAR -------------------------------------- (03)
         USING BINDSECT,R1
         CLC   XBINRET,=X‘00000000‘
         BNE   TERM
         L     R15,EDTADDR
CALLEDT  LA    R13,SAVEAREA -------------------------------------------- (04)
         LA    R1,PLIST ------------------------------------------------ (05)
         BALR  R14,R15 ------------------------------------------------- (06)
         B     LOOP
TERM     TERM
```

```
*
EDTADDR  DC    F‘0‘
SAVEAREA DS    18F
PLIST    DS    0F
OPT1     DS    L1
OPT2     DC    X‘00‘
RESERVE  DC    X‘00‘
F        DC    X‘40‘ --------------------------------------------------- (07)
D        DC    X‘40‘
EDTRES   DS    3C
WRITE    DC    Y(END1-WRITE)
         DC    X‘4040‘
DRSTZ    DC    X‘01‘
TEXT     DC    ,*** OPTION BYTE / "ST" (STOP) ***‘
END1     EQU   *
READ     DS    0L6
LENGTH   DS    L4
MESSG    DS    L2
RDUND    DS    L1
BINDPAR  BIND  SYMBOL=EDT,LIBNAM=$EDTLIB,SYMTYP=CSEN,LDINFO=DEF, -
               SYMBLAD=EDTADDR,MF=L
BINDSECT BIND  MF=D,PREFIX=X
         END
```

(01) Two printable characters are read in using WRTRD.

(02) This PACK instruction converts the printable characters into the desired hexadecimal characters. An input of C’05’ (i.e. X’F0F5’) is placed in field OPT1 as the byte X’05’ by the following MVC.

(03) EDT is linked via the BIND macro to the user program only in the very first EDT call.

(04) The address of the register save area is passed in register 13.

(05) The address of the parameter list, which includes the byte OPT1, is passed in register 1.

(06) This is where the user program branches to EDT. The return address is saved in register 14.

(07) EDT is to set up its data and variable area starting at page X’40’ (decimal 64). The virtual file is to begin at page X’40’ or at the first unoccupied page after X’40’.

```
/assign-syslst to-file=@@.lst.prog.1
/assign-sysdta to-file=*syscmd
/start-program $assemh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '1.1A00' OF '1992-04-30' LOADED
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1990. ALL
RIGHTS RESERVED
%  ASS6010 V1.1A10 OF BS2000 ASSEMBH READY
//compile source=s.prog.1,macro-library=$.macrolib,listing=*parameters
(output=*syslst)
%  ASS6011 ASSEBLY TIME: 440 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 74 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-program $tsoslnk
%  BLS0500 PROGRAM 'TSOSLNK', VERSION 'V21.0E01' OF '1994-01-28' LOADED
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1991. ALL
RIGHTS RESERVED
*program prog1,filenam=c.prog.1,version=166 ---------------------------- (01)
*include prog1,*
*end
%  LNK0500 PROGRAM BOUND
%  LNK0503 PROGRAM FILE WRITTEN: C.PROG.1 ------------------------------ (02)
%  LNK0504 NUMBER PAM PAGES USED:      3
/load-prog c.prog.1
%  BLS0500 PROGRAM 'PROG1',VERSION '166' OF '1996-01-08' LOADED
*** OPTION BYTE / "ST" (STOP) ***05 ------------------------------------ (03)
PROGRAM  EDT V16.6A00  STARTED
   1.     @run (xmpl1,z.lib) ------------------------------------------- (04)
   1.     @print
  56.4321 THIS TASK HAS THE TSN 0444
   1.     edt is now being used
   2.     as a subroutine
   3.     @print
   1.0000 EDT IS NOT BEING USED
   2.0000 AS A SUBROUTINE
  56.4321 THIS TASK HAS THE TSN 0444
   3.     @return
*** OPTION BYTE / "ST" (STOP) ***04 ------------------------------------ (05)
   3.     this is fun
   4.     @return
*** OPTION BYTE / "ST" (STOP) ***02 ------------------------------------ (06)
   4.     @print
   1.0000 EDT IS NOW BEING USED
   2.0000 AS A SUBROUTINE
   3.0000 THIS IS FUN
  56.4321 THIS TASK HAS THE TSN 0444
```

```
   4.     @return
*** OPTION BYTE / "ST" (STOP) ***02
   1.     @print
   1.     @return
*** OPTION BYTE / "ST" (STOP) ***04
   1.     @print
   1.     from the beginning
   2.     @return
*** OPTION BYTE / "ST" (STOP) ***08 ------------------------------------ (07)
   2.     watch this
*** OPTION BYTE / "ST" (STOP) ***08
   3.     @print
   1.0000 FROM THE BEGINNING
   2.0000 WATCH THIS
*** OPTION BYTE / "ST" (STOP) ***10 ------------------------------------ (08)
   3.     oops
*** OPTION BYTE / "ST" (STOP) ***10
   4.     @print
   1.0000 FROM THE BEGINNING
   2.0000 WATCH THIS
*** OPTION BYTE / "ST" (STOP) ***st ------------------------------------ (09)
/
```

(01) The module is to be linked to create a loadable program.

(02) The loadable calling program is written into file C.PROC.1.

(03) Entering 05 places B’0000 0101’ in function byte 1, i.e. bits 2^0 and 2^2 are set:
2^0 must always be set in the first EDT call. 2^2 specifies that EDT is to remain active until it is terminated by means of @RETURN. The current file is not deleted when control is returned to the calling program.

(04) Even if EDT is running as a subroutine, a further subroutine can be called by means of @RUN.
The subroutine XMPL1 is described in the description of @RUN.

(05) Entering 04 places B’0000 0100’ in function byte 1, i.e. bit 2^2 is set.
This ensures that the current file is not deleted when control is returned to the calling program.

(06) Entering 02 places B’0000 0010’ in function byte 1, i.e. bit 2^1 is set.
This specifies that the current file is to be deleted when control is returned to the calling program.

(07) Entering 08 places B’0000 1000’ in function byte 1, i.e. bit 2^3 is set.
This specifies that EDT is to read only a single line and then return control to the calling program. The line which is entered is inserted in the current file.

(08) Entering 10 places B’0001 0000’ in function byte 1, i.e. bit 2^4 is set.
This specifies that EDT is to read only a single line and then return control to the calling program without inserting the entered line in the current file.

(09) Entering the letters ST causes a branch to the end of the program.

Related publications

[1] **EDT V16.6** (BS2000/OSD)
Statements
User Guide

Target group
This manual is intended for established EDT users as well as users not yet familiar with EDT.

Contents
The manual describes the processing of SAM and ISAM files, elements from program libraries and POSIX files. It also contains descriptions of the EDT operating modes, statement codes, procedures and statements.

[2] **EDT V16.6** (BS2000/OSD)
Statement Formats
Ready Reference

Target group
This Ready Reference is intended for EDT users.

Contents
The Ready Reference contains an overview of all EDT statements arranged according to function group and in alphabetical order within groups, together with the statement formats.

[3] **EDT-ARA** (BS2000/OSD)
Additional Information for Arabic
User Guide

Target group
The manual addresses EDT users wishing to edit Arabic texts.

Contens
The manual describes how to create, delete, update, insert and copy bilingual records or parts thereof.
With EDT-ARA it is possible to replace Latin strings with Arabic strings and vice versa, to prefix or suffix records in one script with strings in the other script etc.

[4] **EDT-FAR** (BS2000/OSD)
Additional Information for Farsi
User Guide

Target group
The manual addresses EDT users wishing to edit Farsi texts.

Contens
The manual describes how to create, delete, update, insert and copy bilingual records or parts thereof.
With EDT-FAR it is possible to replace Latin strings with Farsi strings and vice versa, to prefix or suffix records in one script with strings in the other script etc.

[5] **SDF V4.0A** (BS2000/OSD)
Introductory Guide to the SDF Dialog Interface
User Guide

Target group
BS2000/OSD users

Contents
This manual describes the interactive input of commands and statements in SDF format. A Getting Started chapter with easy-to-understand examples and further comprehensive examples facilitates use of SDF. SDF syntax files are discussed.

[6] **BS2000/OSD-BC V2.0A**
Commands, Volume 1, A-L
User Guide

Target group
The manual addresses both nonprivileged BS2000/OSD users and systems support.

Contents
This manual contains BS2000/OSD commands ADD-... to LOGOFF (basic configuration and selected products) with the functionality for all privileges. The introduction provides information on command input.

[7] **BS2000/OSD-BC V2.0**
Commands, Volume 2, M-SG
User Guide

Target group
The manual addresses both nonprivileged users and systems support.

Contents
This manual contains BS2000/OSD commands MODIFY-... to SET-... (basic configuration and selected products) with the functionality for all privileges.

[8] **BS2000/OSD-BC V2.0A**
Executive Macros
User Guide

Target group
The manual addresses all BS2000/OSD assembly language programmers.

Contents
The manual contains a summary of all Executive macros, detailed descriptions of each macro with notes and examples, including job variable macros, and a comprehensive general training section.

[9] **Assembler** (BS2000)
Reference Manual

Target group
Assembly-language users working with BS2000

Contents
– Assembler characteristics
– Assembly language
– Makro language
– Using the assembler
– Messages and error messages
– Flags
– Description of the assembler
– Description of the ADIAG assembler diagnostic program

[10] **ASSEMBH** (BS2000)
User Guide

Target group
Assembly language users under BS2000

Contents
– Calling and controlling ASSEMBH
– Assembling, linking, loading, and starting programs
– Input sources and output of ASSEMBH
– Runtime system, structured programming
– Language interfacing
– Assembler Diagnostic Program ASSDIAG
– Advanced Interactive Debugger AID
– ASSEMBH messages
– Machine instruction formats

[11] **XHCS V1.0**
(BS2000/OSD)
8-Bit Code Processing in BS2000/OSD
User Guide

Target group
Users of the DCAM, TIAM and UTM access methods, system administrators, and users migrating from EHCS to XHCS.

Contents
XHCS (Extended Host Code Support) is a software package of BS2000/OSD that lets you use extended character sets in conjunction with 8-bit terminals. XHCS is also the central source of information on the coded character sets in BS2000/OSD. XHCS replaces EHCS.

[12] **JV V11.2A** (BS2000/OSD)
Job Variables
User Guide

Target group
The manual addresses both nonprivileged users and systems support.

Contents
The manual describes management and possible uses of job variables. The command descriptions are divided according to function areas. The macro calls are described in a separate chapter.

[13] **SDF-P V2.0A** (BS2000/OSD)
Programming in the Command Language
User Guide

Target group
The manual addresses BS2000/OSD users and systems support.

Contents
SDF-P is a structured procedure language in BS2000. The introduction is followed by a detailed description of commands, functions and macros. SDF-P V2.0A can be used under BS2000/OSD-BC V1.0 only with VAS 2.0A and SDF V4.0.

[14] **LMS** (BS2000)
SDF Format
User Guide

Target group
BS2000 users.

Contents
Description of the statements for creating and managing PLAM libraries and the members these contain.
Frequent applications are illustrated with examples.

[15] **POSIX** (BS2000/OSD)
POSIX Basics for Users and System Administrators
User Guide

Target group
BS2000 system administrators, POSIX administrators, BS2000 users, users of UNIX/SINIX workstations.
Contents
This manual describes the following: introduction to and working with POSIX; BS2000 software products in a POSIX environment; installing, controlling and exiting the POSIX subsystem; managing POSIX users via BS2000.

[16] **POSIX** (BS2000/OSD)
Commands
User Guide

Target group
This manual addresses all users of the POSIX shell.
Contents
This manual is designed as a work of reference. It describes working with the POSIX shell and the commands of the POSIX shell in alphabetical order.

Ordering manuals

The manuals listed above and the corresponding order numbers can be found in the Siemens Nixdorf *List of Publications*. New publications are described in the *Druckschriften-Neuerscheinungen (New Publications)*.

You can arrange to have both of these sent to you regularly by having your name placed on the appropriate mailing list. Please apply to your local office, where you can also order the manuals.

Index

Contents

EDT V16.6 (BS2000/OSD)

Subroutine Interface

Target group
This manual is intended for experienced EDT users and programmers.

Contents
The manual describes the subroutine interface of EDT and explains how to call EDT as a subroutine and incorporate the EDT functionality in user-defined programs. It also describes the structure and generation of control blocks, the creation of external statement routines and how to call a user program from within EDT.

Edition: August 1996

File: EDT_UPS.PDF