English

Fujitsu Software BS2000

EDT Unicode Mode

Subroutine Interface

User Guide

Valid for:
EDT V17.0D

Edition June 2022

Comments… Suggestions… Corrections…

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

Feel free to send us your comments by e-mail to: bs2000.info@fujitsu.com.

Certified documentation according to DIN EN ISO 9001:2015

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2015.

Copyright and Trademarks

Table of Contents

EDT Unicode Mode Subroutine Interface

1 Preface

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

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

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

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

  The fact that a Unicode representation is used internally in EDT means that the compatibility of all the various interfaces at which users previously had direct access to the internal EDT data cannot be maintained. This applies to the old L mode subroutine interface, the former @RUN interface and the Locate mode of the IEDTGLE interface. These interfaces can therefore no longer be used in Unicode mode.
- The compatibility mode provides the full functionality of EDT V16.6B with only slight extensions.

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

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

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

1.1 Structure of the EDT documentation

The manuals

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

describe EDT's Unicode mode. The manuals

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

contain a description of the compatibility mode.

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

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

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

1.2 Target groups for the EDT manuals

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

This manual on the EDT subroutine interfaces is intended for experienced EDT users and programmers who want to make use of EDT's extensive capabilities in their own programs.

To call EDT as a subroutine, it is absolutely essential for programmers to possess knowledge of the most important BS2000 commands, be familiar with EDT and the EDT statements and have a good understanding of Assembler and C.

1.3 Structure of the manual “EDT Subroutine Interfaces”

This manual describes only the subroutine interface provided by
EDT V17.0A.

It covers the following topics:

- **Preface**

  Notes on the structure and mode of use of the EDT manuals.
- **New features and changes in EDT V17.0A**

  Summary description of the new features and changes in the EDT V17.0A subroutine interface.
- **Using EDT as a subroutine**

  Description of the functions together with calls and return codes, structure of the control blocks, brief examples.
- **User-defined statements - @USE**

  Presentation of the possibility of writing user-defined statements in EDT. Special application as a statement filter.
- **User routine - @RUN**

  Starting a user routine as a subroutine with the @RUN statement.
- **Production of subroutine interface applications**

  Rules and examples for the production of programs in BS2000 which use EDT as a subroutine or which are to be called by EDT as user routines by means of the @USE or @RUN statements.
  The languages C and Assembler are considered.
- **Extensive sample programs with comments**

  For C and Assembler main programs, C and Assembler user routines.
- **Appendix – C header**

  Layout of the C header file.

References to related documents are provided in abbreviated form in the text. The full title of each publication referenced by a number is provided in full after the corresponding number in the “Related publications” section.

2 Modified and new functionality in EDT V17.0A

The new features and changes in the subroutine interface are described below.

2.1 EDT operating modes

EDT V17.0A can be run in two modes:

- In Unicode mode which has been extended for the processing of Unicode files but in which some older interfaces are no longer supported.
- In compatibility mode which comprises the full functionality of EDT V16.6B but does not offer the functional extensions available in Unicode mode.

2.1.1 Unicode mode

Unicode mode provides extensions for the processing of Unicode files and supports record lengths greater than 256 bytes.
The following are not supported:

- the old L mode subroutine interface
- the @RUN interface in its previous format
- the Locate mode of the `IEDTGLE` interface

Unicode mode provides a new, extended @RUN interface with a modified statement format and a new program interface as well as extensions to the
@UNLOAD and @USE statements in order to support case-sensitive names of up to 32 characters in length for entry points (`ENTRY`).

2.1.2 Compatibility mode

The EDT V16.6B function scope and interfaces are supported in compatibility mode. In particular, the old L mode interface, the @RUN interface in its former format and the Locate mode of the `IEDTGLE` interface are fully supported.

The record length continues to be restricted to 256 bytes.

2.1.3 Interface formats and operating modes

A new format is available for the IEDTGLE interface.

There are therefore two formats:

- V16 format: identical to the IEDTGLE interface in EDT V16.6B
- V17 format: new format with extensions for Unicode

When generating the control blocks for the IEDTGLE interface, the user can choose the required version (see section “Generation and structure of the control blocks”).

Basically, the following applies:

- The IEDTGLE interface (V16 format) is only supported in full by compatibility mode
- The IEDTGLE interface (V17 format) is only supported in full by Unicode mode

To make it possible to develop applications which use the IEDTGLE interface and can run with both compatibility and Unicode mode, there is a new connection module which converts the interfaces. This means that an application which works with the EDT V16.6B IEDTGLE interface can also run unconverted in EDT V17.0A Unicode mode provided that it only uses functions that are also supported by Unicode mode.

If EDT is not already running or if EDT is running but all the work files are empty then the appropriate operating mode is activated when the IEDTCMD interface is called: compatibility mode for V16 format, Unicode mode for V17 format. This mechanism is deactivated as soon as the operating mode is switched explicitly using the @MODE or @CODENAME statement. Any subsequent mode changes must then always be performed explicitly. Equally, an implicit mode change never occurs from within a user routine. Otherwise the return from the user routine to EDT would be unsuccessful.

It should be noted that implicit changes to the operating mode by changing the interface version are only possible via the IEDTCMD interface. Since EDT is in an non-initialized state after such a change, all the function calls which require EDT to be initialized would lead to errors.

In general, it is advisable always to work with the same interface format (V17 format or V16 format) in a program and make any changes of operating mode explicitly (via a @MODE statement).

If EDT is already running and it is not possible to change mode or if mode change is not performed due to a prior explicit change (see above) the interface format is converted.

- A call of the IEDTGLE interface in V16 format is converted to V17 format if EDT is running in Unicode mode.
- A call of the IEDTGLE interface in V17 format is converted to V16 format if EDT is running in compatibility mode.

This conversion is only possible if the employed function is supported by the currently active EDT operating mode. For example, the V16 format of the AMCB cannot be converted into V17 format if Locate mode is required. In this case, the return code EAMPAERR/EAMPA08 is returned.

If the IEDTGLE interface is used in V16 format then:

- Locate mode cannot be converted.
- When the global status is read (IEDTGET function with pseudo work file ‘G’), the field for the globally specified character set which is omitted in V17 format is only set to a value other than ‘blank’ if the same character set is specified in all the non-empty work files.

In the case of the IEDTGLE interface in V17 format, the subset that can be converted into V16 format is defined (*compatible* V17 format). This is specified in the description in each case.

The full scope of the V17 format (*extended* V17 format) can only be used with Unicode mode.

To ensure that extensions are not used unintentionally, it is possible to specify at the V17 format interface that only the compatible format may be used (by setting the EGLCOMP flag). In this case, the use of the functions of the extended format is rejected with a return code (also in Unicode mode).

Using the extended format (EGLCOMP flag not set) also prevents the automatic conversion of the V17 format into the V16 format even in cases where this is possible. However, it should be noted that the flag is a *conversion* flag and not a *switchover* flag. This means that even in the case of a compatible format, there may be a *switchover* to Unicode mode if this is possible.

V16 format is identical to the format of the IEDTGLE interface in EDT V16.6B Consequently, an application which uses the *compatible* V17 format can also run with EDT V16.6B if it includes the EDT V17.0A connection module IEDTGLE.

The overview indicates the possible combinations:

	EDT V17 not present	**EDT V17 present, switchover permitted and call via IEDTCMD**	**EDT V17 present, compatibility mode active, switchover prohibited**	**EDT V17 present, Unicode mode active, switchover prohibited**
Call via IEDTGLE V16 interface	As previously	Set compatibility mode	No special action necessary	Convert to V17 – if not possible: error
Call via IEDTGLE V17 interface (compatible format)	Convert to V16 interface	Set Unicode mode	Convert to V16 interface	No special action necessary
Call via IEDTGLE V17 interface (extended format)	Error	Set Unicode mode	Error	No special action necessary
Call via L mode interface	As previously	Set compatibility mode	No special action necessary	Error

2.2 Long records

In Unicode mode, it is possible to process records of up to 32768 **characters** in length. At the `IEDTGLE` interface, it is possible to transfer records of up to 32768 **bytes** in length (as in the DMS).

This does not require any extension of the `IEDTGLE` interface. This also applies to the `IEDTGLE` interface in V16 format. The user must make sure that the buffer is sufficiently long.

2.3 Local character sets

In Unicode mode, it is possible to set different character sets for the individual work areas. As a result, the character set information is no longer transferred when the global settings are read but instead when the work file-specific settings are read. The corresponding field is no longer located in the control block EDTPARG but in the control block EDTPARL.

2.4 Locate mode

The Locate mode of the `IEDTGLE` interface is not supported in EDT V17.0A Unicode mode.

2.5 Memory organization

In EDT V17.0A Unicode mode, users cannot influence EDT memory organization. This is unnecessary since Locate mode is not supported and users no longer have direct access to records in EDT memory.

2.6 L mode interface

The old L mode interface (predecessor of the `IEDTGLE` interface) is not supported in EDT V17.0A Unicode mode.

2.7 @RUN interface

EDT V17.0A Unicode mode provides a new @RUN interface with a modified statement format and a new, extended program interface.

2.8 @UNLOAD statement

In Unicode mode, the @UNLOAD statement has been extended by operands which make it possible to unload a complete BLS load unit (`UNIT`).

2.9 @USE statement

In Unicode mode, the @USE statement has been extended by operands which make it possible to specify a case-sensitive name of up to 32 characters in length for entry points (ENTRY).

2.10 User-defined statements and user routines in high-level languages

User-defined statements declared via the @USE interface and user routines called via the @RUN interface can be written in high-level languages (e.g. C) provided that they support ILCS linkage. It may be necessary to activate ILCS linkage via a compiler option.

2.11 Empty records

In Unicode mode, EDT V17.0A is able to read and write records of length 0.

When using V16 format, it is also possible to read and write records of length `0` if EDT is running in Unicode mode.

2.12 Interfaces and their interactions

EDT provides three interfaces:

- An interface at which a user program calls EDT functions (`IEDTGLE` interface)
- An interface at which routines written by a user are declared as user-defined statements. The associated routine is called in order to execute this type of user-defined statement (@USE interface).
- An interface for calling user routines (@RUN)

These three interfaces use the same control block layout.

The @USE interface also makes it possible to declare a statement filter:
the declared routine is called for each statement in order to determine whether or not the statement can be executed.

For their part, routines which implement user-defined statements as well as user routines can call EDT functions via the `IEDTGLE` interface (sometimes with restrictions).

3 Using EDT as a subroutine

The interface for calling EDT consists of the following components:

- a connection module (`IEDTGLE`) with multiple entry point addresses, and
- a series of control blocks which are generated with Assembler macros or C includes.

When a function is called, the corresponding entry address is called and the addresses of the required control blocks and buffers are passed to it as parameters. The number and type of the parameters are different for each function. For each function, the fields in the passed control blocks that are to be evaluated and for which entries must be supplied are specified (see sections “Statement functions” and “Logical record access functions”).

3.1 Linking the user program to EDT

The IEDTGLE module from the module library is linked in the user program (main program) via an external reference (e.g. a V constant).

The IEDTGLE module

- contains all the entry point addresses for use of the individual functions
- calls the dynamically or preloadable component of EDT using the BIND macro
- saves the EDT entry point address in the global control block EDTGLCB

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

The IEDTGLE module is reentrant. IEDTGLE is used to branch to EDT. If the call is performed using the compatible V17 interface format and no V17 EDT is installed in the current system, IEDTGLE converts the V17 format (if this is both possible and permitted) into the V16 format. Otherwise, the parameters are passed to EDT unchanged.

3.1.1 Calling EDT

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

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

PARAMETERLIST

Users must create this data area themselves.

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

The parameter list is dependent on the function of the call. The descriptions of the individual functions indicate the parameters that need to be specified (see section “Statement functions” and “Logical record access functions”).

SAVEAREA

Register save area (18 words, `DC 18F'0'`) which must be created by the caller. EDT saves the registers here.

RETURN

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

ENTRY

The `IEDTGLE` module contains a separate entry point address for each function:

Entry point	Function
IEDTINF	Read the EDT version number
IEDTCMD	Execute an EDT statement sequence
IEDTEXE	Execute an EDT statement sequence
IEDTGET	Read a record Read the global status Read the local status of a work file
IEDTGTM	Read a marked record
IEDTPUT	Write a record
IEDTPTM	Mark a record

IEDTDEL	Delete a record range Delete the copy buffer
IEDTREN	Modify the line number

3.2 Generation and structure of the control blocks

The section below describes the macros for the generation of the control blocks, the structure of the control blocks and the meaning of the fields in the control blocks.

The description adheres to the structure of the corresponding Assembler macro. The layout of the interfaces in C (header files) is described in section “Appendix - C header files”.

Any fields which are not described are intended for internal use by EDT. Users may not modify their contents.

The following overview indicates the assignment of the control blocks to the V16 and V17 formats:

Control block	V16 format	V17 format
EDTGLCB	Version 1	Version 2
EDTUPCB	Version 2	Version 3
EDTAMCB	Version 1	Version 2
EDTPARG	Version 1	Version 2
EDTPARL	Versions 1, 2, 3	Version 4

On a call, the employed control blocks must either all belong to the V16 format or all belong to the V17 format. Calls containing a mixture of control blocks in V16 and V17 format are rejected.

3.2.1 EDTGLCB - Global EDT control block

EDTGLCB represents the global control block in all EDT program interfaces. It is used by the IEDTGLE interface as well as by the @USE and @RUN interface.

Creating the EDTGLCB control block

The IEDTGLCB Assembler macro is used to create the EDTGLCB control block.

Name	Operation	Operands
[name]	IEDTGLCB	[{ D | C }] [,prefix] [,VERSION= { 1 | 2 }]

name – Symbolic name of the first DS statement when C is specified.
– DSECT name if D is specified.

If *name* is not specified then EDTGLCB is used (preceded by *prefix* if
specified).

D A dummy section (DSECT) is generated.

C A memory section with symbolic addresses is generated (no CSECT

statement).

prefix A character with which the generated field names should start.

If *prefix* is not specified, E is used by default.

VERSION Selects which version of the control block is to be generated:

Version 1 is used with the V16 format of the interface.
Version 2 is used with the V17 format of the interface.

If the macro IEDTGLCB VERSION=2 is specified then the EDTGLCB control block is generated in the following form:

```
          IEDTGLCB D,VERSION=2
1 EDTGLCB  MFPRE DNAME=EDT,MF=D
2 EDTGLCB  DSECT ,
2               *,##### PREFIX=I, MACID= #####
1 *--------------- EDT UNIT NUMBER, EDTGLCB VERSION NUMBER --------------
1 EGLUNITC EQU   66                EDT UNIT NUMBER
1 EGLVERSC EQU   2                 EDTGLCB VERSION NUMBER
1 EGLVERSL EQU   12                VERSION-LENGTH (INFO)
1 EGLMSGM  EQU   80                MAX LENGTH FOR MESS
1 *--------------- EDT MAIN-RETURNCODES --------------------------------
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 EUPCMPER EQU   X'0030'           COMPATIBILITY VIOLATION       V17.0
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 SUB-RETURNCODE1 ---------------------------------
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 EUPPA20  EQU   X'14'              ERROR IN CCSN                 V17.0
1 EUPPA24  EQU   X'18'              CONVERSION ERROR              V17.0
1 *                    *------ MAIN: EUPVEERR -------------------------
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 (NO LONGER USED)
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 EAMPA40  EQU   X'28'              ERROR IN CCSN                 V17.0
1 EAMPA44  EQU   X'2C'              CONVERSION ERROR              V17.0
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(XGLUNITC)      UNIT NUMBER
1          DS    AL1                RESERVED
1 EGLVERS  DC    AL1(XGLVERSC)      FUNCTION INTERFACE VERSION NUMBER
1 *                      *--- RETURN CODE ----------------------------
1 EGLRETC  DS    0A                 GENERAL RETURN CODE
1 EGLSRET  DS    0AL2               SUB RETURN CODE
1 EGLSR2   DC    AL1(0)             SUB RETURN CODE2
```

```
1 EGLSR1   DC    AL1(0)             SUB RETURN CODE1
1 EGLMRET  DC    AL2(0)             MAIN RETURN CODE
1 *                       *---- RETURN MESSAGE FIELD --------------------
1 EGLINFM  DS    0F                 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 EGLCOMP  EQU   X'80'              COMPATIBLE/EXTENDED FORMAT    V17.0
1 EGLILCS  EQU   X'40'              ILCS ENVIRONMENT              V17.0
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 FILENAME
1 *                       *---- USER PARAMETERS -------------------------
1 EGLUSR1  DC    XL4'00000000'      USER PARAMETER1 (EDT CALLER    )
1 EGLUSR2  DC    XL4'40404040'      USER PARAMETER2 (SUBROUTINE    )
1 EGLUSR3  DC    XL4'40404040'      USER PARAMETER3 (EXIT ROUTINE  )
1 *                       *---- CHARACTER SET --------------------------
1 EGLCCSN  DC    CL8' '             CODED CHARACTER SET NAME      V17.0
1 EGLIND2  DC    X'00'              INDICATOR BYTE 2              V17.0
1 EGLCMOD  EQU   X'80'              COMP MODE RUNNING             V17.0
1 EGLRES   DC    X'000000'          RESERVED                      V17.0
1 *--------------- LENGTH OF CONTROL BLOCK -----------------------------
1 EGLGLCBL EQU   *-EDTGLCB
```

Meaning of the control block fields		Length (bytes)	Format	Parameter type	
				Call	Return
EGLUNIT	Unique identification of EDT.	2	X	A (M)	
EGLVERS	Control block version number.	1	X	C (M)	
EGLSR1	SUBCODE1: Subvalue of the return code, unique within the main value.	1	X		R
EGLSR2	SUBCODE2: Subvalue of the return code, unique within the main value.	1	X		R

EGLMRET	MAINCODE: Main value of the return code. The individual return codes are explained in more detail on "EDTGLCB - Global EDT control block "ff.	2	X		R
EGLCMDS/ EGLINFM	In the case of a syntax error, this field contains the offset of the invalid statement from the start of the statement sequence (IEDTCMD and IEDTEXE functions). In the case of the IEDTINF function, EDT always enters 0 for the EGLINFM field.	4	X		R
EGLRMSGL	Length of the message in EGLRMSGF. If no message is passed then the field has the value zero.	2	X	C	R
EGLRMSGF	EDT passes an (error) message to the user program in this field.	80	P	C	R
EGLCDS	In this field, EDT informs an application filter of the send key with which the statement was transmitted in F mode dialog. EGLDUE DUE key EGLF1 F1 function key EGLF2 F2 function key EGLF3 F3 function key EGLK1 K1 function key For the values for other function keys, see [18].	1	X		R
EGLINDB	The indicator byte contains flags with various meanings.	1			
Flag EGLCOMP	The function call should be restricted to the compatible format: *not set* (default): The extended V17 format is used. *set*: The compatible V17 format is used. See also section 2.1.3 (Interface formats and operating modes).			C	
Flag EGLILCS	This flag (*ILCS* flag) should be set if EDT is called from within a C main program (see section 6.1 (Producing main programs in C) onwards). If EDT V16 is called with the compatible V17 format then this flag is ignored.			C	
Flag EGLSPL	This flag (*SPL* flag) must be set if EDT is called from within an SPL environment.			C	

Flag EGLSTXIT	This flag must be set if the EDT interrupt routines are to be activated during the call. It is only activated if the call is performed via the IEDTCMD and IEDTEXE interfaces.			C	
Flag EGLINIT	This flag is set by EDT after initialization and is deleted when EDT terminates.			D (M)	R
Flag EGLDTVD	This flag is set by EDT after initialization and is deleted when EDT terminates.			D (M)	
Flag EGLETVD	This flag is set by EDT after initialization. It continues to be set after EDT has terminated.			D (M)	
EGLFILE	EDT enters the number of the current work file ($0..$22) in this field (left-aligned) on a return from the function IEDTCMD. The remainder of the field is padded with blanks. This also applies to EDTGLCB for which a statement or user routine must be specified.	8	P		R
EGLUSR1	In this field, the caller of the IEDTCMD interface can specify a value which EDT makes available unchanged to other users of the IEDTGLE or @USE interface (see below for details). The field is prefilled with binary zeros.	4		C	R
EGLUSR2	In this field, a user of the @USE interface can specify a value which EDT makes available unchanged to other users of the @USE or IEDTGLE interface (see below for details). The field is prefilled with blanks.	4		C	R
EGLUSR3	This field is not currently taken over by EDT. The content is therefore undefined.	4			
EGLCCSN	Name of the character set (possibly padded with blanks or terminated with X'00') in which certain parameters are encoded: COMMAND, MESSAGE1, MESSAGE2 and EDTREC (on reading and writing). See the sections on the statement and record access functions. If no character set is specified (eight characters or first character binary zero), then the character set is determined in accordance with the rules set out below.	8	P	C	
EGLIND2	This field currently contains the EGLCMOD flag.	1			

Flag EGLCMOD	EDT sets this flag if EDT is running in compatibility mode, that is to say if the control blocks have been converted back from V16 format.				R

C	Call parameter	Must be supplied by the caller.
(M)		This is set by the macro (when the P parameter is set) and should not be changed by the user.
D	Call parameter	Must be deleted by the caller on the first call and may not be subsequently modified.
R	Return parameter	Supplied by EDT.
X	Binary format	Binary digits
P	Printable	Printable texts (in the character set EDF03IRV).

Character set selection if there is no specification in EGLCCSN

For the IEDTCMD and IEDTEXE functions:

1. If a global character set has been set in EDT, that is to say if the same character set has been set for all the non-empty EDT work files, and this character set is a 7-bit or 8-bit EBCDIC character set then it is used.
2. If 1) does not make it possible to determine a character set then the character set for the current work file is used provided that this is a 7 or 8-bit EBCDIC character set.
3. If neither 1) nor 2) makes it possible to determine a character set then the EDF041 character set is used.

For the IEDTGET, IEDTGTM and IEDTPUT functions:

1. If the work file which is being read from or written to is not empty then this work file's character set is used.
2. If the work file is empty then the same procedure as for IEDTCMD and IEDTEXE is used.

Treatment of the user parameters

EDT manages global instances of the user parameters EGLUSR1, EGLUSR2 and EGLUSR3. It takes over values from a GLCB supplied by the user into these global instances in the following cases:

1. The value of EGLUSR1 is taken over if the IEDTCMD interface is called by a user program.
2. The value of EGLUSR2 is taken over on return to EDT from a user routine (@RUN) or a statement or filter routine (@USE). This also applies on returns from the associated initialization routines.
3. At present, the value of EGLUSR3 is never taken over.

EDT enters the values of all three global instances in a GLCB in the following cases:

1. EDT calls a user routine (@RUN) or a statement or filter routine (@USE). This also applies on calls of the associated initialization routines.
2. EDT returns to a user program after an IEDTCMD or IEDTEXE call.

Changes compared to the V16 format

- The EGLREOR flag is no longer used.
- The EGLCOMP flag is new.
- The EGLILCS flag is new.
- The EGLCCSN field is new.
- The EGLIND2 field in combination with the EGLMOD flag is new.
- The return codes EUPCMPER, EUPPA20, EUPPA24, EAMPA40 and EAMPA44 are new.

Compatible V17 format: the EGLCCSN field may only contain blanks.

3.2.2 EDTUPCB - Subroutine control block

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

Creating the EDTUPCB control block

The IEDTUPCB Assembler macro is used to create the EDTUPCB control block.

Name	Operation	Operands
[name]	IEDTUPCB	[{ D | C }] [,prefix] [,VERSION= { 2 | 3 }]

name – Symbolic name of the first DS statement when C is specified.
– DSECT name if D is specified.

If *name* is not specified then EDTUPCB is used (preceded by *prefix* if
specified).

D A dummy section (DSECT) is generated.

C A memory section with symbolic addresses is generated (no CSECT

statement).

prefix One character with which the generated field names should start.

If *prefix* is not specified, E is used by default.

VERSION Selects which version of the control block is to be generated:

Version 2 is used with the V16 format of the interface.
Version 3 is used with the V17 format of the interface.

If the macro IEDTUPCB VERSION=3 is specified then the EDTUPCB control block is generated in the following form:

```
          IEDTUPCB D,VERSION=3
1 EDTUPCB  MFPRE DNAME=EDT, MF=D
2 EDTUPCB  DS    0F
1 *--------------- EDT UNIT NUMBER, EDTUPCB VERSION NUMBER --------------
1 EUPUNITC EQU   66                  EDT UNIT NUMBER
1 EUPVERSC EQU   3                   EDTUP VERSION NUMBER
1 EUPCMDM  EQU   (32763+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  DC    AL2(XUPUNITC)       UNIT NUMBER
1          DS    AL1                 RESERVED
1 EUPVERS  DC    AL1(XUPVERSC)       FUNCTION INTERFACE VERSION NUMBER
1          DS    A                   RESERVED
1 *                      *---- INHIBIT FLAGS ---------------------------
1 EUPINHBT DC    X'00'               INHIBIT FLAG BYTE
1 EUPMODE  EQU   X'80'               * NO SWITCH TO COMPATIBLE MODE V17.0
1 EUPNTXT  EQU   X'40'               * NO <TEXT>     (HALT / RETURN)
1 EUPN@EDO EQU   X'20'               * NO @EDIT ONLY (L-MODE : RDATA)
1 EUPN@EDT EQU   X'10'               * NO @EDIT      (L-MODE : WRTRD)
1 EUPNUSER EQU   X'08'               * NO USER-PROG. (@RUN/@USE)
1 EUPNBKPT EQU   X'04'               * NO BKPT (@SYSTEM)
1 EUPNCMDM EQU   X'02'               * NO CMD (@SYSTEM <STRING>)
1 EUPNEXEC EQU   X'01'               * NO MEXEC/MLOAD (@EXEC/@LOAD)
1 EUPNINHB EQU   X'00'               * NO RESTRICTIONS
1          DS    AL3                 RESERVED
1 *--------------- LENGTH OF CONTROL BLOCK ------------------------------
1 EUPUPCBL EQU   *-EDTUPCB
```

Meaning of the control block fields		Length (bytes)	Format	Parameter type	
				Call	Return
EUPUNIT	Unique identification of EDT.	2	X	C(M)	
EUPVERS	Control block version number.	1	X	C(M)	

EUPINHBT	By setting the individual bits, it is possible to disable certain statements for users in order to prevent undefined termination of EDT or prevent an exit from Unicode mode: EUPMODE Disables switchover between Unicode mode and compatibility mode EUPNTXT Disallows to specify message in @HALT and @RETURN EUPN@EDO Disables @EDIT ONLY EUPN@EDT Disables @EDIT EUPNUSER Disables @USE and @RUN EUPNBKPT Disables the @SYSTEM statement (without operands) EUPNCMDM Disables the @SYSTEM statement (with operands) EUPNEXEC Disables the @EXEC / @LOAD statement	1		C	

C	Call parameter	Must be supplied by the caller.
(M)		This is set by the macro (when the P parameter is set) and should not be changed by the user.
R	Return parameter	Supplied by EDT.
X	Binary format	Binary digits

Changes compared to the V16 format

The EUPMODE flag is new.

Compatible V17 format

- The EUPMODE flag must not be set.

- The constant `EUPMSGM` has only been retained for reasons of compatibility. The maximum length of the message is determined dynamically depending on the terminal type.

3.2.3 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 required to access the work files.

Creating the EDTAMCB control block

The Assembler macro IEDTAMCB is used to generate the EDTAMCB control block.

Name	Operation	Operands
[name]	IEDTAMCB	[{ D | C }] [,prefix] [,VERSION= { 1 | 2 }]

name – Symbolic name of the first DS statement when C is specified.
– DSECT name if D is specified.

If *name* is not specified then EDTAMCB is used (preceded by *prefix* if
specified).

D A dummy section (DSECT) is generated.

C A memory section with symbolic addresses is generated (no CSECT

statement).

prefix One character with which the generated field names should start.

If *prefix* is not specified, E is used by default.

VERSION Selects which version of the control block is to be generated:

Version 1 is used with the V16 format of the interface.
Version 2 is used with the V17 format of the interface.

If the macro IEDTAMCB VERSION=2 is specified then the EDTAMCB control block is generated in the following form:

```
          IEDTAMCB D,VERSION=2
1 EDTAMCB  MFPRE DNAME=EDT,MF=D
2 EDTAMCB  DSECT ,
2               *,##### PREFIX=I, MACID= #####
1 *--------------- EDT UNIT NUMBER, EDTAMCB VERSION NUMBER --------------
1 EAMUNITC EQU   66                EDT UNIT NUMBER
1 EAMVERSC EQU   2                 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  DC    AL2(XAMUNITC)     UNIT NUMBER
1          DS    AL1               RESERVED
1 EAMVERS  DC    AL1(XAMVERSC)     FUNCTION INTERFACE VERSION NUMBER
1          DS    A                 RESERVED
1          DS    X                 RESERVED                       V17.0
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' '            FILENAME
1 EAMDISP  DC    F'0'              DISPLACEMENT
1 EAMLKEY1 DC    H'8'              LENGTH OF KEY1
1 EAMLKEY2 DC    H'8'              LENGTH OF KEY2
1 *                     *--- INPUT PARAMETERS (ONLY IN MOVE MODE) ------
1 EAMPKEY  DC    H'8'              LENGTH OF KEY OUTPUT BUFFER
1 EAMPREC  DC    H'0'              LENGTH OF RECORD OUTPUTBUFFER
1 *                     *--- INPUT / OUTPUT PARAMETERS -----------------
1 EAMLKEY  DC    H'8'              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 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          DS    AL2               RESERVED
1 *--------------- LENGTH OF CONTROL BLOCK -----------------------------
1 EAMAMCBL EQU   *-EDTAMCB
```

Meaning of the control block fields	Length (bytes)	Format	Parameter type	
			Call	Return

EAMUNIT	Unique identification of EDT.	2	X	C(M)	
EAMVERS	Control block version number.	1	X	C(M)	
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 are to be marked (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 C C	
EAMFILE	The work file variable ($0..$22) specifying the work file (0-22) which is to be accessed, or the values G or L0 to L22 when the work file status is queried, or the value C when the copy buffer is deleted.	8	P	C	
EAMDISP	This field contains: –the relative position of the required record when a record is read (IEDTGET) –the required search direction when a marked record is read (IEDTGTM).	4	X	C	
EAMLKEY1	Length of the EDTKEY1 buffer	2	X	C	
EAMLKEY2	Length of the EDTKEY2 buffer	2	X	C	
EAMPKEY	The length of the output buffer for the line number must be entered in this field before a record is read (IEDTGET, IEDTGTM). Currently, only the value 8 is permitted.	2	X	C	
EAMPREC	The length of the output buffer for the record must be entered in this field before a record is read (IEDTGET, IEDTGTM).	2	X	C	
EAMLKEY	Length of the transferred data in the EDTKEY buffer. On a read operation, EDT passes the actual length of the read record number; on a write operation, the user passes the length of the record number to be written. Currently, only the value 8 is permitted.	2	X	C	R

EAMLREC	Length of the transferred data in the EDTREC buffer. On a read operation, EDT passes the actual length of the read record; on a write operation, the user passes the length of the record to be written.	2	X	C	R
EAMMARK	Specifies the marking of a record. It is used for both input and output. The mark field is used as additional information for each field. The user may use the marks 1..9 (EAMMK01..EAMMK09) as well as the marks 13, 14 and 15 EAMMK13, EAMMK14 and EAMMK15) in IEDTPUT and IEDTPTM as desired. The other marks are reserved for special functions.	2		C	R

C	Call parameter	Must be supplied by the caller.
(M)		This is set by the macro (when the P parameter is set) and should not be changed by the user.
R	Return parameter	Supplied by EDT.
X	Binary format	Binary digits
P	Printable	Printable texts (in the character set EDF03IRV).

Changes compared to the V16 format

- The EAMMMODB field and the two flags EAMMOVM and EAMLOCM are no longer used.
- The equates for the unused marks (EAMMK10, EAMMK11, EAMMK12 and EAMMK0) are no longer used.

Compatible V17 format

No restrictions.

3.2.4 EDTPARG - Global parameter settings

After the global parameter settings have been read with IEDTGET, EDT saves the information in the EDTPARG control block.

Creating the EDTPARG control block

The Assembler macro IEDTPARG is used to generate the EDTPARG control block.

Name	Operation	Operands
[name]	IEDTPARG	[{ D | C }] [,prefix] [,VERSION= { 1 | 2 }]

name – Symbolic name of the first DS statement when C is specified.
– DSECT name if D is specified.
If *name* is not specified then EDTPARG is used (preceded by *prefix* if specified).

D A dummy section (DSECT) is generated.

C A memory section with symbolic addresses is generated (no CSECT statement).

prefix One character with which the generated field names should start.
If *prefix* is not specified, E is used by default.

VERSION Selects which version of the control block is to be generated:
Version 1 is used with the V16 format of the interface.
Version 2 is used with the V17 format of the interface.

If the macro IEDTPARG VERSION=2 is specified then the EDTPARG control block is generated in the following form:

```
          IEDTPARG D,VERSION=2
1 EDTPARG  MFPRE DNAME=EDT,MF=D
2 EDTPARG  DSECT ,
2               *,##### PREFIX=I, MACID= #####
1 *--------------- EDT UNIT NUMBER, EDTPARL VERSION NUMBER --------------
1 EPGUNITC EQU   66                  EDT UNIT NUMBER
1 EPGVERSC EQU   2                   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(XPGUNITC)       UNIT NUMBER
1          DS    AL1                 RESERVED
1 EPGVERS  DC    AL1(XPGVERSC)       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          DS    XL10                RESERVED
1 *--------------- TOTAL LENGTH OF CONTROL BLOCK ------------------------
1 EPGPARGL EQU   *-EDTPARG
```

Meaning of the control block fields		Length (bytes)	Format	Parameter type	
				Call	Return
EPGUNIT	Unique identification of EDT.	2	X	C(M)	
EPGVERS	Control block version number.	1	X	C(M)	
EPGMODE	Current mode: F = F mode L = L mode C = Caller (i.e. from a statement which is processed via the IEDTGLE interface).	1	P		R
EPG@SYM	EDT statement symbol	1	P		R
EPGWDS1	Window size 1 (2..24)	2			R
EPGWDS2	Window size 2 (0.0..22)	2			R
EPGFIL1	Work file in first window ($0..$22) padded with blanks	8	P		R

EPGFIL2	Work file in second window ($0..$22) padded with blanks	8	P		R

C	Call parameter	Must be supplied by the caller.
(M)		This is set by the macro (when the P parameter is set) and should not be changed by the user.
R	Return parameter	Supplied by EDT.
X	Binary format	Binary digits
P	Printable	Printable texts (in the character set EDF03IRV).

Changes compared to the V16 format

The EPGCCSN field is no longer used (it now has the name EPLCCSN in the control block EDTPARL). When the IEDTGET (PARG) interface is called, it is no longer necessary to enter a value in the EAMPREC field in AMCB.

When EDT V17.0A is called in Unicode mode with a version 1 EDTPARG control block, the EPGCCSN field is only set to a value other than blank if the same character set has been specified for all non-empty work files.

Compatible V17 format

No restrictions.

3.2.5 EDTPARL - Work file-specific parameter settings

When reading the work file-specific parameter settings with the IEDTGET function, EDT saves the information in the EDTPARL control block.

Creating the EDTPARL control block

The Assembler macro IEDTPARL can be used to generate the EDTPARL control block.

Name	Operation	Operands
[name]	IEDTPARL	[{ D | C }] [,prefix] [,VERSION= { 1 | 2 | 3 | 4 }]

name – Symbolic name of the first DS statement when C is specified.
– DSECT name if D is specified.

If *name* is not specified then EDTPARL is used (preceded by *prefix* if
specified).

D A dummy section (DSECT) is generated.

C A memory section with symbolic addresses is generated (no CSECT

statement).

prefix One character with which the generated field names should start.

If *prefix* is not specified, E is used by default.

VERSION Selects which version of the control block is to be generated:

Versions 1, 2, 3 are used with the V16 format of the interface.
Version 4 is used with the V17 format of the interface.

If the macro IEDTPARL VERSION=4 is specified then the EDTPARL control block is generated in the following form:

```
          IEDTPARL D,VERSION=4
1 EDTPARL  MFPRE DNAME=EDT,MF=D,PREFIX=*
2 EDTPARL  DSECT ,
2               *,##### PREFIX=, MACID= #####
1 *--------------- EDT UNIT NUMBER, EDTPARL VERSION NUMBER --------------
1 EPLUNITC EQU   66                EDT UNIT NUMBER
1 EPLVERSC EQU   4                 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(XPLUNITC)     UNIT NUMBER
1          DS    AL1               RESERVED
1 EPLVERS  DC    AL1(XPLVERSC)     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 IF EBCDIC
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          DS    CL4               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    XL1024            COMMON AREA FOR FILE DESCRIPTION
1 EPLOPEND EQU   *                 END OF COMMON AREA
1          ORG   XPLOPENC          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
1          ORG   XPLOPENC          DESCRIPTION OF OPENED UFS FILE
1 EPLOPNX  DS    CL1024            NAME OF OPENED UFS FILE
1 *
1 EPLCCSN  DS    CL8               CODED CHARACTER SET NAME       V17.0
1 EPLCCSNG DS    XL1               CCS IS GLOBAL (0/1)            V17.0
1 EPLSSTRU DS    H                 STRUCTURE SYMBOL UTF16
1 *--------------- LENGTH OF CONTROL BLOCK -----------------------------
1 EPLPARLL EQU   *-EDTPARL
```

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

Meaning of the control block fields		Length (bytes)	Format	Parameter type	
				Call	Return
EPLUNIT	Identification of EDT.	2	X	C(M)	
EPLVERS	Control block version number.	1	X	C(M)	
EPLVPOS	1st line in the data window (00000001..99999999), same value as EPLVPOS1	8	C		R
EPLHPOS	1st column in the data window (1..32768), same value as EPLHPOS1)	2	X		R
EPLRLIM	Max. record length in F mode (1..32768)	2	X		R
EPLINF	Information On/Off (1/0)	1	P		R
EPLLOW	Lower On/Off (1/0)	1	P		R
EPLHEX	Hex On/Off (1/0)	1	P		R
EPLEDL	Edit Long On/Off (1/0)	1	P		R
EPLSCALE	Scale On/Off (1/0)	1	P		R
EPLPROT	Protection On/Off (1/0)	1	P		R
EPLSTRUC	Structure symbol (if EBCDIC coding, if another coding is used then the value is simply stored in EPLSSTRU)	1	C		R
EPLOPEN	OPEN display = R: ISAM real P: PLAM I: ISAM virtual S: SAM virtual X: UFS/POSIX 0: No file opened	1	P		R
EPLEMPTY	File empty yes/no (1/0)	1	P		R
EPLMODIF	File modified yes/no (1/0)	1	P		R
EPLSTDF	Default file name set with @FILE	54	P		R
EPLSTDL	Default library name set with @PAR LIBRARY=...	54	P		R

EPLSTDT	Default type set with @PAR ELEMENT-TYPE=...	8	P		R
EPLOPNFL	EPLOPEN = R: ISAM file name, real, opened P: PLAM library name I: ISAM file name S: SAM file name, padded with blanks	54	P		R
EPLOPNE	EPLOPEN = P: PLAM element name, padded with blanks	64	P		R
EPLOPNV	EPLOPEN = P: PLAM version number, padded with blanks	24	P		R
EPLOPNT	EPLOPEN = P: PLAM type, padded with blanks	8	P		R
EPLOPNX	EPLOPEN = X: POSIX file name, padded with blanks, not ending with 0	1024	P		R
EPLVPOS1	First line number in window 1	8	P		R
EPLHPOS1	First column in window 1	2	X		R
EPLVPOS2	First line number in window 2	8	P		R
EPLHPOS2	First column in window 2	2	X		R
EPLINDX1	Index Off/On/Full window 1 (0/1/2), as set with @PAR INDEX=...bzw. @PAR EDIT-FULL=...	1	P		R
EPLINDX2	Index Off/On/Full window 2 (0/1/2)	1	P		R
EPLCCSN	Character set for the work file. If the work file is empty, EPLCCSN contains blanks.	8	P		R
EPLCCSNG	The character set applies to all non-empty work files Yes/No (1/0)	1	P		R
EPLSSTRU	Structure symbol in UTF16	2	U		R

C Call parameter Must be supplied by the caller.

(M) This is set by the macro (when the P parameter is set) and should not be changed by the user.

D	Call parameter	Must be deleted with binary zeros by the user on the first call.
R	Return parameter	Supplied by EDT.
X	Binary format	Binary digits
P	Printable	Printable texts (in the character set EDF03IRV).
U	UTF16 characters	Text in UTF16 coding

Changes compared to the V16 format

- The EPLCCSN field is new (it previously had the name EPGCCSN in the control block EDTPARG).
- The EPLCCSNG field is new.
- The EPLSTCOD field is no longer used.
- The EPLOPNXC field is no longer used.
- The EPLSTRUC field now only contains the structure symbol if this can be displayed as an EBCDIC character; otherwise it contains binary zeros. The EPLSSTRU field always contains the structure symbol in UTF16 coding.

Compatible V17 format

If V17 format is used to call EDT in compatibility mode or to call an EDT version lower than V17.0 then the information items not supplied by the relevant V16 version are set to default values:

- All file name fields with blanks
- EPLVPOS1 and EPLVPOS2 with '00010000'
- EPLHPOS1 and EPLHPOS2 with binary 1
- EPLINDX1 and EPLINDX2 with '1'
- EPLCCSN with the globally specified CCSN (EPLCCSNG then contains 1) if the work file is not empty; otherwise with blanks (EPLCCSNG then contains 0).

3.3 Buffers

The employed buffer names (`EDTREC`, `EDTKEY`, `EDTKEY1`, `EDTKEY2`, `COMMAND`) are only placeholders for the purposes of the description. Users can choose the buffer names freely in their programs.

If multiple buffers are used in a function then they must be coded in the samecharacter set.

3.3.1 Record (EDTREC)

The buffer is used

- to transfer the record via EDT when a record is read (`IEDTGET`, `IEDTGTM` functions) or
- to transfer the record to EDT when a record is written (`IEDTPUT` function).

The minimum length of a record is 0 bytes and the maximum length 32768 bytes and there is *no* record length field. The length is specified in the `EDTAMCB` control block.

3.3.2 Line number (EDTKEY, EDTKEY1, EDTKEY2)

These buffers contain a line number.

A line number is 8 bytes in length. The permitted range for line numbers in awork file goes

from	`C'00000001'`	(hexadecimal `'F0F0F0F0F0F0F0F1'`)
to	`C'99999999'`	(hexadecimal `'F9F9F9F9F9F9F9F9'`)

In order to address the first or last record in a work file, the input parameters can also be specified in binary form as follows in the access functions `IEDTGET`, `IEDTGTM` and `IEDTDEL`:

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

Only values from the specified range may be entered for the buffers `EDTKEY1`, `EDTKEY2` and `EDTKEY`. The user must enter the value 8 in the associated length fields in `EDTAMCB`. Any specifications outside of this range return an error code.

3.3.3 Sequence of statements in a buffer (COMMAND)

In the functions IEDTCMD and IEDTEXE, the buffer contains a sequence of statements that are to be executed by EDT.

When a user-defined statement or a user routine is called, the buffer contains the text which was specified at call time.

If a statement filter is called, the buffer contains the statement to be filtered.

It has the following format: 2 bytes for the length of COMMAND (total length including that of the length field and the unused field), 2 bytes unused, n bytes record content.

The maximum length is 32767 bytes (4+32763).

3.3.4 Messages in a buffer (MESSAGE1, MESSAGE2)

These buffers all contain a message that is to be output for the `IEDTCMD` function.

They have the following format: 2 bytes for the length of `MESSAGE` (total length including that of the length field and the unused field), 2 bytes unused, n bytes record content.

The length field must contain the length in bytes. However, a maximum of 132 or 80 characters are displayed (depending on the current line length of the screen).

This is now only unsatisfactorily reproduced by the *Equate* `EUPMSGM` function which is now only supported for reasons of compatibility.

3.4 Statement functions

The statement functions are used to:

- select an EDT version
- query information about the EDT version number
- pass a statement or sequence of statements to EDT

The following functions are available:

Functions	Entry point address
Read the EDT version number or select the EDT version	`IEDTINF`
Execute EDT statements	`IEDTCMD`
Execute EDT statements without a screen dialog	`IEDTEXE`

3.4.1 IEDTINF - Read EDT version number

At the IEDTINF interface, users can see the version number of the loaded EDT. If no EDT is as yet loaded then it is now loaded but not initialized. If EDT is installed under IMON then it is possible to specify the desired EDT version.

The EDT version for dynamic loading is specified in the EGLRMSGF field (printable). The length of the version specification must be entered in the EGLRMSGL field.

Format of the version number: `EDT Vaa.a[d[ii]]` where a and i are digits and d is a letter

A version specification is recognized on the basis of the length entry in the EGLRMSGL field which must have a value of between 9 and 12 to be valid. If no version is specified and multiple versions co-exist then the version set using /SELECT-PRODUCT-VERSION or the highest EDT version is dynamically loaded.

If dynamic loading of the specified version is not possible then the return code EUPVEERR is set in the EGLMRET field and EDT is not loaded. If it is possible to identify the STD version then this is entered in the EGLRMSGF field. If it is not possible to determine the STD version then the sub-return code EUPVE04 is entered in the EGLSR1 field. If the call is successful, the return code EUPRETOK is set in the EGLMRET (EDTGLCB) field.

EDT passes the loaded EDT version in the format described above in the EGLRMSGF field (independently of whether loading was performed at call time or EDT was already loaded).

If EDT is already loaded, then the version of the loaded EDT is always returned even if the caller has specified another version. In this case, the return code is again EUPRETOK.

Call

The following specifications are required:

- Completion of the required fields in the EDTGLCB control block
- Call of the IEDTINF entry point address with the parameter list

Overview

(For the control block, see section “EDTGLCB - Global EDT control block”).

Entry point address	:	IEDTINF

Parameter list	:	A(EDTGLCB)

Call parameter		Return parameter	
EDTGLCB:	EGLUNIT EGLVERS EGLINDB EGLRMSG	EDTGLCB:	EGLRETC EGLRMSG EGLINFM

Note

In Unicode mode, EDT always passes 0 in the EGLINFM field.

Return codes

EGLMRET	EGLSR1
EUPRETOK EUPEDERR EUPPAERR EUPVEERR	EUPOK00 00 EUPPA04 EUPVE00 EUPVE04

The fields EGLMRET and EGLRS1 belong to the control block EDTGLCB. For the meaning of the return codes, see section “EDTGLCB - Global EDT control block”.

Call in the C program

Required include files:

```
#include <stdio.h>

#include <iedgle.h>
```

The EDTGLCB control block is declared and initialized as follows:

```
iedglcb glcb = IEDGLCB_INIT;
```

In the C program, the IEDTINF function is called as follows:

```
IEDTINF(&glcb);
```

3.4.2 IEDTCMD - Execute EDT statements

This call passes a statement or sequence of statements to EDT for execution.
If the application field (length 4) is empty then processing returns immediately to the calling program.

The following statements are permitted at the IEDTCMD interface.

@+	@GETVAR	@SEQUENCE
@-	@HALT	@SET
@:	@INPUT (Format 1, 2)	@SETF
@AUTOSAVE	@LIMITS	@SETJV
@BLOCK	@LIST	@SETLIST
@CHECK (Format 2)	@LOAD	@SETSW
@CLOSE	@LOG	@SETVAR
@CODENAME	@LOWER	@SHOW
@COLUMN	@MODE	@SORT
@COMPARE	@MOVE	@STAJV
@CONVERT	@ON	@STATUS
@COPY	@OPEN	@SUFFIX
@CREATE (Format 1+ 2)	@P-KEYS	@SYMBOLS
@DELETE	@PAGE	@SYNTAX
@DELIMIT	@PAR	@SYSTEM
@DIALOG	@PREFIX	@TABS
@DO (Format 1)	@PRINT	@TMODE
@DROP	@QUOTE	@UNLOAD
@EDIT (Format 1, 2, 3)	@RANGE	@UNSAVE
@ELIM	@READ	@USE
@ERAJV	@RENUMBER	@VDT
@EXEC	@RESET	@VTCSET
@FILE	@RETURN	@WRITE
@FSTAT	@RUN	@XCOPY
@GET	@SAVE	@XOPEN
@GETJV	@SEARCH-OPTION	@XWRITE
@GETLIST	@SEPARATE	

The user-defined statements are also permitted (see chapter “User defined statements -@USE”).

The @EDIT statement (except with format 4 - @EDIT LONG...) is always interpreted as @EDIT ONLY at the `IEDTCMD` interface and causes a switchover to line mode dialog.

The EDT statement symbol does not have to be specified (except in the case of `@:`).

The program run (initialization, transition to user dialog, termination with unload and release of memory) is controlled by means of the sequence of statements passed to EDT.

Once EDT has been loaded, its data area is initialized (on the 1st call only).

After executing the sequence of statements, EDT returns to the calling program.

@HALT in the passed sequence of statements results in the termination of EDT (release of memory and unloading).

If the statement sequence does not end with @HALT then control returns to the calling program without the data area being released. Processing can be continued by issuing a new call with a statement sequence or @HALT can be specified to terminate EDT.

In a routine which processes a user-defined statement or in a user routine (see section "User defined statements - @USE" and "User routines - @RUN"), it is only possible to call the IEDTCMD interface if the call is intended to address an instance of EDT other than the calling instance.

If statements addressed to the calling EDT instance use the global control block EDTGLCB passed to the statement routine then they are permitted only via the IEDTEXE interface. A call of the IEDTCMD interface from a statement routine with the EDTGLCB of the calling instance is rejected with the return code EUPPAERR/EUPPA08. For further information, see chapter "User defined statements - @USE" and following sections.

If an error occurs (syntax or runtime error) then execution is immediately interrupted with a corresponding return code and an error message. In such cases, the EGLCMDS (in EDTGLCB) field is used as an error pointer. This points to the start of the invalid statement within the statement sequence. For reasons of compatibility, the first character after the record length field is numbered '1' (using this numbering convention, the first character in the passed statement sequence has the number '3'). Counting is always performed in characters not in bytes. The return code EUPSYERR or EUPRTERR is passed.

User dialog

It is possible to switch to the user dialog by means of the @DIALOG statement (screen dialog) or by means of @EDIT ONLY (line mode dialog) in the passed statement sequence.

Whenever processing switches to the screen dialog as the result of @DIALOG in the passed statement sequence, the transferred messages (MESSAGE1, MESSAGE2) are displayed in the message lines.

@EDIT ONLY switches to line mode dialog (read with RDATA).

The user dialog is terminated with @END, @HALT or @RETURN or, in F mode,by pressing the [K1] key.

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

If <message> is specified in a @HALT or @RETURN statement then the message text is also entered in the EGLRMSGF message field of the EDTGLCB control block.

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

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

The flag EGLSTXIT in EDTGLCB is evaluated on every call via the IEDTCMD interface. On return to the calling program, the EDT interrupt routines are exited (if they have been requested).

If the flag EUPNUSER is set in EDTUPCB then attempts to execute a @USE statement are rejected in the dialog.

Control structures

The following data areas must be defined before calling the function:

- the control block EDTGLCB
- the control block EDTUPCB
- the statement sequence (COMMAND)
- 2 optional message lines (MESSAGE1 and MESSAGE2), otherwise a null pointer

The control blocks can be found in the section "Generation and structure of the control blocks". The COMMAND, MESSAGE1 and MESSAGE2 buffers are described in the section on buffers.

If the screen is not split, MESSAGE1 is displayed in the message line in the dialog. If the screen is split, MESSAGE1 is displayed in the first and MESSAGE2 in the second message line.

If the length field of MESSAGE1 or MESSAGE2 has a value smaller than or equal to 4 then the corresponding message is not output. If this occurs in the first call to the user dialog, the EDT start message is output.
The same applies if a null pointer is called specified.

If @DIALOG or @EDIT ONLY occurs other than as the last statement in the statement sequence then it should be noted that the return code and message in EDTGLCB may be due to subsequent statements.

Call

The following specifications are required (see overview):

- Entry of values in the required fields in the EDTGLCB and EDTUPCB control blocks.
- Entry of the statement sequence in the COMMAND buffer.
- Entry of the message texts in the MESSAGE1 and MESSAGE2 or entry of null pointers in the corresponding fields in the parameter list.
- Call of the entry point address IEDTCMD with the parameter list

Overview

(For the control blocks, see section "EDTGLCB - Global EDT control block").

```
Entry point address      :     IEDTCMD
```

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

Call parameter		Return parameter	
EDTGLCB:	EGLUNIT EGLVERS EGLINDB EGLCCSN	EDTGLCB:	EGLRETC EGLRMSG EGLCMDS EGLFILE
EDTUPCB:	EUPUNIT EUPVERS EUPINHBT		
COMMAND MESSAGE1 / NULL MESSAGE2 / NULL			

Note
On each return, the return code and the name of the current work file (EGLFILE) are entered in the EDTGLCB control block.

Return codes

EGLMRET	EGLRS1
EUPRETOK	EUPOK00 EUPOK04 EUPOK08 EUPOK12 EUPOK16 EUPOK20
EUPSYERR EUPRTERR EUPEDERR EUPOSERR EUPUSERR EUPPAERR	00 00 00 00 00 EUPPA04 EUPPA08 EUPPA12 EUPPA16 EUPPA20 EUPPA24
EUPSPERR EUPABERR	00 EUPOK08

EGLMRET and EGLRS1 are fields in the control block EDTGLCB. For the meaning of the return codes, see section “EDTGLCB - Global EDT control block”.

Example

```
************************************************************
* CMDBSP:  EXAMPLE OF EXECUTION OF AN EDT STATEMENT SEQUENCE    *
*          IN UNICODE MODE                                      *
*          (PAR SPLIT=OFF,LOWER=ON,SCALE=ON,INDEX=ON;DIALOG)    *
************************************************************
*
CMDBSP   START
CMDBSP   AMODE ANY
CMDBSP   RMODE ANY
         BALR  R10,0
         USING *,R10
         MVC   EGLCCSN,CCSN041
         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,VERSION=2
         IEDTUPCB C,VERSION=3
* - 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'DIALOG END WITH 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)
*
CCSN041  DC    CL8'EDF041 '
*
         END   CMDBSP
```

Call in the C program

Required include files:

```
#include <stdio.h>

#include <iedtgle.h>
```

The control blocks `EDTGLCB` and `EDTUPCB` are declared and initialized as follows:

```
iedglcb glcb = IEDGLCB_INIT;

iedupcb upcb = IEDUPCB_INIT;
```

For the format and values to be entered in the structures `command`, `message1` and`message2`, refer to the example in section “Example 1 - C main program”.

The `IEDTCMD` function is called with the addresses of these structures:

```
IEDTCMD(&glcb,&upcb,&command,&message1,&message2);
```

3.4.3 IEDTEXE - Execute EDT statements without screen dialog

This call passes a statement or sequence of statements to EDT for execution.

It differs from the IEDTCMD function as follows:

- EDT must already be loaded and initialized.
- It is not possible to conduct a screen dialog (@DIALOG and @EDIT are not permitted).
- EDT cannot be terminated or unloaded (@HALT, @RETURN and @MODE are not permitted).
- It is not possible to start any EDT procedures (@INPUT and @DO are not permitted).

The following statements are permitted at the IEDTEXE interface.

@+	@LIST	@SETJV
@-	@LOAD	@SETLIST
@BLOCK	@LOG	@SETSW
@CHECK (Format 2)	@LOWER	@SETVAR
@CLOSE	@MOVE	@SHOW
@CODENAME	@ON	@SORT
@COLUMN	@OPEN	@STAJV
@COMPARE	@P-KEYS	@STATUS
@CONVERT	@PAGE	@SUFFIX
@COPY	@PAR	@SYMBOLS
@CREATE (Format 1+ 2)	@PREFIX	@SYSTEM
@DELETE	@PRINT	@TABS
@DELIMIT	@QUOTE	@TMODE
@ELIM	@RANGE	@UNLOAD
@ERAJV	@READ	@UNSAVE
@EXEC	@RENUMBER	@USE
@FILE	@RESET	@VDT
@FSTAT	@SAVE	@VTCSET
@GET	@SEARCH-OPTION	@WRITE
@GETJV	@SEPARATE	@XCOPY
@GETLIST	@SEQUENCE	@XOPEN
@GETVAR	@SET	@XWRITE
@LIMITS	@SETF	

Unlike in EDT V16.6, the flag EGLSTXIT in EDTGLCB is evaluated on every call via the IEDTEXE interface. On return to the calling program, the EDT interrupt routines are exited (if they have been requested). If the calling program is a statement or user routine then the status of interrupt handling is restored to its state before the call to the external routine when control returns to EDT.
If a syntax or runtime error occurs then execution is immediately interrupted with a corresponding return code and an error message. In the case of a syntax error, the EGLCMDS field (EDTGLCB) is used as an error pointer. This points to the start of the invalid statement within

the statement sequence. For reasons of compatibility, the first character after the record length field is numbered '1' (using this numbering convention, the first character in the passed statement sequence has the number '3'). Counting is always performed in characters not in bytes. The return code EUPSYERR or EUPRTERR is passed.

Unlike in the case of the IEDTCMD interface, the IEDTEXE interface may also be used in a routine which executes a user-defined statement or in a user routine (see the section on user-defined statements - @USE and user routines - @RUN) for statements sent to the calling EDT instance.

If the IEDTEXE function is called from the statement routine in a user-defined statement then no further user-defined statements may be entered.

Control structures

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

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

For a description of the control block EDTGLCB, see section "EDTGLCB - Global EDT control block". For a description of the COMMAND buffer, see section "Sequence of statements in a buffer (COMMAND)".
The control block EDTGLCB which is passed by EDT should be used in the statement routine in a user-defined statement.

Call

The following specifications are required (see overview):

- Entry of values in the control block fields in EDTGLCB
- Entry of the statement sequence in the COMMAND data field.
- Call of the IEDTEXE entry point address with the parameter list

Overview

(For the control blocks, see section "Generation and structure of the control blocks").

Entry point address	:	IEDTEXE

Parameter list	:	A (EDTGLCB, COMMAND)

Call parameter		Return parameter	
EDTGLCB:	EGLUNIT EGLVERS EGLINDB EGLCCSN	EDTGLCB:	EGLRETC EGLRMSG EGLCMDS EGLFILE EGLUSR1 EGLUSR2 EGLUSR3
COMMAND			

Return codes

EGLMRET	EGLRS1

EUPRETOK	00
EUPSYERR	00
EUPRTERR	00
EUPEDERR	00
EUPOSERR	00
EUPUSERR	00
EUPPAERR	EUPPA04 EUPPA12 EUPPA20

The fields EGLMRET and EGLRS1 are fields in the control block EDTGLCB.
For the meaning of the return codes, see section “EDTGLCB - Global EDT control block”.

Call in the C program

Required include files:

```
#include <stdio.h>

#include <iedtgle.h>
```

The function IEDTEXE is also called in the C program with the address of EDTGLCB and the command for execution:

```
IEDTEXE(&glcb,&command);
```

3.5 Logical record access functions

The logical record access functions allow users to access records in the work files from within a user program.

The following access functions are available:

Access functions	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 line number	IEDTREN
Read the global EDT parameter settings	IEDTGET
Read the work file-specific EDT parameter settings	IEDTGET

The individual access functions are described in the sections below.

Access functions can only be executed if EDT is initialized. Initialization is performed by calling the function IEDTCMD (see section “IEDTCMD - Execute EDT statements”).

If the call is issued in a statement or user routine with the supplied EDTGLCB, then EDT is already initialized.

If access functions are used with a work file which is already active and is being processed as a procedure with @DO then they are rejected with the return code EAMACERR/EAMAC48 (work file is active).

Processing files and library elements

The logical record access functions always apply to the records in a work file. If it is necessary to process files or library elements then these must first be read into a work file (e.g. with the function IEDTCMD). A file can also be opened for real processing. These files or elements are then written back using EDT statements (e.g. with the function IEDTCMD).

Each record has a line number via which it can be accessed.
Each record may also have various marks (see [1], section on record marks).

Control structures

Before a record access function is called, the required control blocks must have been declared in the calling program and, if necessary, defined and filled with the required values.

Return codes of the record access functions

The table indicates the return codes which EDT can set for the individual access functions. In this table, GET stands for IEDTGET etc.

Return code	Function

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

	EAMPA32 EAMPA36 EAMPA40 EAMPA44	x x x x	x x x x	x x x x	x x	x x	x x

The fields EGLMRET and EGLRS1 belong to the control block EDTGLCB. For the meaning of the return codes, see section “EDTGLCB - Global EDT control block”.

3.5.1 IEDTGET - Read a record

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

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

- Work file (`$0..$22`) in the field `EAMFILE` (`EDTAMCB`): work file from which the record is to be read
- Line number of a work file record in the buffer `EDTKEY1` (the record does not have to exist).
- Displacement n (`0`, `+N`, `-N`) in the field `EAMDISP` (`EDTAMCB`): distance (in records) from the specified line number in binary format
- Character set in which the record is to be made available, in the field `EGLCCSN` (`EDTGLCB`)

Records with record mark 13 (see section “IEDTPTM - Mark a record”) are only taken into account if the flag `EAMIGN13` is set in the field `EAMFLAG` in the control block `EDTAMCB`. Otherwise, they are handled as if they did not exist.

The interface permits two types of addressing:

- Absolute addressing
- Relative addressing

Reading a record with a specific line number – absolute addressing

Displacement specification (`EAMDISP`): `n = 0`

If the value `0` is specified for the displacement then the record with the corresponding line number (`EDTKEY1`) is searched for.

If this record is not present then the record with the next line number is passed (possibly the first or last record).

Summary of the return codes and the read record:

EDTKEY1	EAMDISP	EGLMRET	EAMSR1	EDTREC
Line number of an existing record	`0`	`EAMRETOK`	`EAMOK00`	Record with line number = `EDTKEY1`
Smaller than line number of first record	`0`	`EAMRETOK`	`EAMOK08`	First record
Larger than line number of last record	`0`	`EAMRETOK`	`EAMOK12`	Last record

Larger than line number of first record and smaller than line number of last record and not the line number of an existing record	0	EAMRETOK	EAMOK04	Next record after EDTKEY1
Work file does not contain any records		EAMACERR	EAMAC16	

If the search is successful, EDT passes the line number of the record that was actually read in EDTKEY.
For the meaning of the return codes, see section “EDTGLCB - Global EDT control block”.

Reading a record with relative addressing

Displacement specification (EAMDISP): n = +N/-N (Nî0)

The address of the record to be read consists of

- the line number of a record (EDTKEY1),
- the displacement N of the record relative to the specified line number:

  +N: the Nth (logical) record after the specified line number is read,

  -N: the Nth (logical) record before the specified line number is read,

If the required record lies outside of the work file's line number range then the first or last record is returned.

If either of the line numbers X'0000000000000000' or X'FFFFFFFFFFFFFFFF' is specified as the number of the record that is to be read then the displacement specification is calculated relative to a fictitious record occurring either in front of or after all the others. I.e. the displacement specification 1 or -1 then returns either the first or the last record in the work file.

Summary of the return codes and the read record:

EDTKEY1	EAMDISP	EGLMRET	EAMSR1	EDTREC
EDTKEY1 = XXXXXXXX	+N (N <= records after EDTKEY1)	EAMRETOK	EAMOK00	Record N after EDTKEY1
EDTKEY1 = XXXXXXXX	-N (N <= records before EDTKEY1)	EAMRETOK	EAMOK08	Record N before EDTKEY1
EDTKEY1 = XXXXXXXX	+N (N > records after EDTKEY1)	EAMRETOK	EAMOK12	Last record

EDTKEY1 = XXXXXXXX	-N (N > records before EDTKEY1)	EAMRETOK	EAMOK04	First record
Work file does not contain any records		EAMACERR	EAMAC16	

If the search is successful, EDT passes the line number of the record that was actually read in EDTKEY.

For the meaning of the return codes, see section “EDTGLCB - Global EDT control block”.

Call

The following specifications are required (see overview):

- Entry of values in the required fields in the EDTGLCB and EDTAMCB control blocks.
- Entry of values in the EDTKEY1 buffer
- Provision of storage space for the buffers EDTKEY and EDTREC
- Call of the entry point address IEDTGET with the parameter list

Overview

(For the control blocks, see section “Generation and structure of the control blocks”).

```
Entry point address        :     IEDTGET
```

```
Parameter list             :     A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY,
                                    EDTREC)
```

Call parameter		Return parameter	
EDTGLCB:	EGLUNIT EGLVERS EGLCCSN	EDTGLCB:	EGLRETC EGLRMSG
EDTAMCB:	EAMUNIT EAMVERS EAMFILE EAMDISP EAMLKEY1 EAMPKEY EAMPREC	EDTAMCB:	EAMMARK EAMLKEY EAMLREC
EDTKEY1 EDTKEY EDTREC		EDTKEY EDTREC	

For the possible return codes, see “Logical record access functions”.

Return parameters on successful record access

Alongside the EGLRETC field in EDTGLCB (EGLMRET = EAMRETOK), EDT specifies the following parameters:

Record	EDTREC
Record length	EAMLREC in EDTAMCB
Line number	EDTKEY (always 8 bytes in length)
Length of line number	EAMLKEY in EDTAMCB (always 8 bytes)
Record mark	EAMMARK in EDTAMCB

If the buffer length (EAMPREC) is not sufficient to accommodate the record, EAMACERR is entered for EGLMRET and EAMAC12 is entered for EGLSR1. The record is truncated. The actual length read is entered in the EAMLREC field in AMCB.

In compatibility mode (as in EDT V16.6), the buffer length required to read the record in full is stored here. In Unicode mode, however, it is necessary to specify the number of bytes actually transferred since multi-byte coding always requires the record to be truncated between two valid characters and, consequently, fewer bytes may be transferred than are specified in EAMPREC.

If access is unsuccessful, the fields EAMLKEY and EAMLREC are filled with the value 0.

Call in the C program

Required include files:

```
#include <stdio.h>

#include <iedtgle.h>
```

The control block EDTAMCB is declared and initialized as follows (the value 1024 here was selected as an example of the maximum expected record length):

```
iedamcb amcb = IEDAMCB_INIT;
char rec[1024];
char key[8], key1[8];
amcb.length_key1 = 8;
amcb.length_key_outbuffer = 8;
amcb.length_rec_outbuffer = 1024;
```

The specifications for the other parameters are user-dependent. If, for example, the1st record in work file 0 is searched for:

```
strncpy(amcb.filename,"$0     ",8);
strncpy(key1,"00000001",8);
amcb.displacement = 0;
```

Function call:

```
IEDTGET(&glcb,&amcb,key1,key,rec);
```

3.5.2 IEDTGTM - Read a marked record

This access function makes it possible to search for a marked record starting at a specific line number (EDTKEY1). The direction of the search can also be specified.

It is only possible to search for marked records. It is not possible to search for a specific mark.

Searches can be performed in the work files 0..22. No marked records can be read in a work file in which a file has been opened for real processing by means of @OPEN (format 2). The access attempt is rejected with a return code.

Searching for a marked record

In order to search for a marked record it is necessary to specify the following:

- Work file ($0..$22) in the field EAMFILE (EDTAMCB):
  Work file in which the marked record is to be read.
- Line number of a file record in the buffer EDTKEY1 (the record does not have to exist).
- Displacement n (0,+1,-1) in the field EAMDISP (EDTAMCB):
  Specification of the search direction (other positive values for displacement are handled like +1, and other negative values are handled like -1)
- Character set in which the record is to be made available, in the field EGLCCSN (EDTGLCB)

Two types of search are possible:

- Search for a specific line number
- Search for the next marked record relative to a specified line number

Reading a marked record with a specific line number

Displacement specification: n = 0

If the value 0 is specified for the displacement (EAMDISP) then the record with the specified line number (EDTKEY1) is read. If this record is not present or if the record has no marks then the next marked record (possibly the first or last record) is read and transferred.

Summary of the return codes and the read record:

EDTKEY1	EAMDISP	EGLMRET	EAMSR1	EDTREC
Line number of an existing record	0	EAMRETOK	EAMOK00	Record with record number
Smaller than line number of first marked record	0	EAMRETOK	EAMOK08	First record with mark
Larger than line number of last marked record	0	EAMRETOK	EAMOK12	Last record with mark

Larger than line number of first marked record and smaller than line number of last marked record and not a line number of a marked record	0	EAMRETOK	EAMOK04	Next marked record after EDTKEY1
Work file is empty or contains no marked records		EAMACERR	EAMAC16	

If the search is successful, EDT passes the line number of the record that was actually read in EDTKEY together with the record mark in EAMMARK (EDTAMCB).

For the meaning of the return codes, see section “EDTGLCB - Global EDT control block”.

Reading the next marked record

Displacement specification n = +1 or -1

The next marked record before or after a specific line number is searched for (EDTKEY1).

n = +1: the first marked record after the specified line number is read

n = -1: the first marked record before the specified line number is read

If no further marked records are present in the specified direction then the first or last marked record is read and transferred.

Summary of the return codes and the read record:

EDTKEY1	EAMDISP	EGLMRET	EAMSR1	EDTREC
EDTKEY1 = XXXXXXXX	+1 Record with mark exists after EDTKEY1	EAMRETOK	EAMOK00	Next record with mark after EDTKEY1
EDTKEY1 = XXXXXXXX	-1 Record with mark exists before EDTKEY1	EAMRETOK	EAMOK00	Next record with mark before EDTKEY1
EDTKEY1 = XXXXXXXX	+1 No record with mark after EDTKEY1	EAMRETOK	EAMOK12	Last marked record

EDTKEY1 = XXXXXXXX	-1 No record with mark before EDTKEY1	EAMRETOK	EAMOK08	First marked record
Work file is empty or contains no marked records		EAMACERR	EAMAC16	

If the search is successful, EDT passes the line number of the record that was actually read in EDTKEY together with the record mark in EAMMARK (EDTAMCB).

For the meaning of the return codes, see section “EDTGLCB - Global EDT control block”.

Call

The following specifications are required (see overview):

- Entry of values in the required fields in the EDTGLCB and EDTAMCB control blocks.
- Entry of values in the EDTKEY1 buffer
- Provision of storage space for the buffers EDTKEY and EDTREC
- Call of the entry point address IEDTGTM with the parameter list

Overview

(For the control blocks, see section “EDTGLCB - Global EDT control block”).

```
Entry point address      :     IEDTGTM
```

```
Parameter list           :     A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY,
                                  EDTREC)
```

Call parameter		Return parameter	
EDTGLCB:	EGLUNIT EGLVERS EGLCCSN	EDTGLCB:	EGLRETC EGLRMSG
EDTAMCB:	EAMUNIT EAMVERS EAMFLAG EAMFILE EAMDISP EAMLKEY1 EAMPKEY EAMPREC	EDTAMCB:	EAMMARK EAMLKEY EAMLREC
EDTKEY1 EDTKEY EDTREC		EDTKEY EDTREC	

Return parameters on successful record access:

Alongside the EGLRETC field in EDTGLCB (EGLMRET = EAMRETOK), EDT specifies the following parameters:

Record	EDTREC
Record length	EAMLREC in EDTAMCB
Line number	EDTKEY (always 8 bytes in length)
Length of line number	EAMLKEY in EDTAMCB (always 8 bytes)
Record mark	EAMMARK in EDTAMCB

If the buffer length (EAMPREC) is not sufficient to accommodate the record, EAMACERR is entered for EGLMRET and EAMAC12 is entered for EGLSR1. The record is truncated. The actual length read is entered in the EAMLREC field in AMCB. In compatibility mode (as in EDT V16.6B), the buffer length required to read the record in full is stored here.

In Unicode mode, however, it is necessary to specify the number of bytes actually transferred since multi-byte coding always requires the record to be truncated between two valid characters and, consequently, fewer bytes may be transferred than are specified in EAMPREC.

Call in the C program

Required include files:

```
#include <stdio.h>

#include <iedtgle.h>
```

The control block EDTAMCB is declared and initialized as follows (the value 4096 here was selected as an example of the maximum expected record length):

```
iedamcb amcb = IEDAMCB_INIT;
char key1[8];
char key[8];
char rec[4096];
ambc.length_key1 = 8;
amcb.length_key_outbuff = 8;
amcb.length_rec_outbuff = 4096;
```

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

```
strncpy(amcb.filename,"$22     ",8);
strncpy(key1,"01234000",8);
amcb.displacement = 1;
```

In the C program, the IEDTGTM function is called as follows:

```
IEDTGTM(&glcb,&amcb,key1,key,rec);
```

3.5.3 IEDTPUT - Write a record

This access function saves a record (EDTREC) with a mark (EAMMARK in EDTAMCB) in a work file under the specified line number (EDTKEY).

If a record with this line number already exists then it is replaced together with its mark (see section “IEDTPTM - Mark a record”).

The character set in which the record is made available must be specified in the field EGLCCSN (EDTGLCB).

If the flag EAMNOMOD is set in the EAMFLAG field of control block EDTAMCB when a record is written with IEDTPUT, then the work file is not identified as modified even though a record has been added. This makes it easier for the calling program to recognize whether a user has modified the work file in the dialog (by querying the field EPLMODIF in the control block EDTPARL). This also eliminates an unnecessary EDT save query in the @HALT statement if nothing has been changed in the dialog.

Call

The following specifications are required (see overview):

- Entry of values in the required fields in the EDTGLCB and EDTAMCB control blocks.
- Entry of values in the EDTKEY buffer
- Entry of values in the EDTREC buffer
- Call of the entry point address IEDTPUT with the parameter list

Overview

(For the control blocks, see section “Generation and structure of the control blocks”).

Entry point address	:	IEDTPUT

Parameter list	:	A (EDTGLCB, EDTAMCB, EDTKEY, EDTREC)

Call parameter		Return parameter	
EDTGLCB:	EGLUNIT EGLVERS EGLCCSN	EDTGLCB:	EGLRETC EGLRMSG
EDTAMCB:	EAMUNIT EAMVERS EAMFLAG EAMFILE EAMMARK EAMLKEY EAMLREC		
EDTKEY EDTREC			

For the possible return codes, see section “Logical record access functions”.

Call in the C program

Required include files:

```
#include <stdio.h>

#include <iedtgle.h>
```

The control block `EDTAMCB` is declared and initialized as follows (the value `4096` here was selected as an example of the maximum expected record length):

```
iedamcb amcb = IEDAMCB_INIT;
char key[8];
char rec[4096];
IEDAMCB_SET_NO_MARKS(amcb);
amcb.length_key = 8;
```

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

```
strncpy(rec,"This is the content of the record",29);
strncpy(key,"00750000",8);
strncpy(amcb.filename,"$0     ",8);
amcb.length_rec = 29;
```

In the C program, the `IEDTPUT` function is called as follows:

```
IEDTPUT(&glcb,&amcb,key,rec);
```

3.5.4 IEDTPTM - Mark a record

This access function can be used to mark a record in a work file or delete its mark.

No records can be marked in a work file in which a file has been opened for real processing by means of @OPEN (Format 2).

Possible record marks:

- The record marks 1 to 9 (EAMMK01 to EAMMK09 in EDTAMCB) are available to users without restriction. They can modify these marks using the functions
  IEDTPUT (write record and mark) and IEDTPTM (write record mark).
  These marks can also be set or deleted using statements (statement codes).
- Record mark 13 (EAMMK13 in EDTAMCB) has the special function of an ignore indicator. Records marked in this way are
  - automatically deleted on return from the user dialog (initiated with @DIALOG or @EDIT ONLY) (see section “IEDTCMD - Execute EDT statements”)
  - not included when writing to a file or library element
  - not copied when lines are copied
  - only taken into account by the record access functions IEDTGET and IEDTPTM if the flag EAMIGN13 is set in the field EAMFLAG in the control block EDTAMCB.
- Record mark 14 (EAMMK14 in EDTAMCB) has the special function of an update indicator. Records marked in this way can be overwritten in F mode even if this is not explicitly requested by the dialog user.
- Record mark 15 (EAMMK15 in EDTAMCB) has the special function of a write protection indicator. Records with record mark 15 are write-protected. In the F mode screen dialog, they cannot be set to overwritable by means of the statement code X or [F2].

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

Record marks 13, 14 and 15 can only be modified by the record access functionsIEDTPUT and IEDTPTM but not by means of EDT statements.
Modifying the record in the dialog (i.e. entering data in the data window) deletes record markers 13, 14 and 15.

The line number of the record to be marked must be specified in the EDTKEY field.

Marks are set as a logical OR operation. If all the bits (including the undefined bits) in EAMMMARK are equal to zero then the mark is deleted.

If there is no record with the specified record number then the call is rejected with a return code.

Call

The following specifications are required (see overview):

- Entry of values in the required fields in the EDTGLCB and EDTAMCB control blocks.
- Entry of values in the EDTKEY buffer
- Call of the entry point address IEDTPTM with the parameter list

Overview

(For the control blocks, see section “Generation and structure of the control blocks”).

Entry point address	:	IEDTPTM

Parameter list	:	A (EDTGLCB, EDTAMCB, EDTKEY)

Call parameter		Return parameter	
EDTGLCB:	EGLUNIT EGLVERS	EDTGLCB:	EGLRETC EGLRMSG
EDTAMCB:	EAMUNIT EAMVERS EAMFLAG EAMFILE EAMMARK EAMLKEY		
EDTKEY			

For the possible return codes, see section “Logical record access functions”.

Call in the C program

Required include files:

```
#include <stdio.h>

#include <iedtgle.h>
```

The EDTAMCB control block is declared and initialized as follows:

```
iedamcb amcb = IEDAMCB_INIT;
char key[8];
IEDAMCB_SET_NO_MARKS(amcb);
amcb.length_key = 8;
```

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

```
strncpy(amcb.filename,"$1     ",8);
strncpy(key,"01234000",8);
MARK_14(amcb) = 1;
```

In the C program, the IEDTPTM function is called as follows:

```
IEDTPTM(&glcb,&amcb,key);
```

3.5.5 IEDTDEL - Delete a record range or the copy buffer

This access function deletes either the specified record range of an existing work file or the copy buffer.

Deleting a record range has the same effect as the @DELETE statement (format 1)

Deleting the copy buffer has the same effect as the statement code * in F mode.

Deleting a record range

The record range is specified by two line numbers:

EDTKEY1: Line number of the first record in the range

EDTKEY2: Line number of the last record in the range

If the value X'0000000000000000' is specified in EDTKEY1 and the value X'FFFFFFFFFFFFFFFF' is specified in EDTKEY2 then all the records in the work file are deleted.
This corresponds to the statement @DELETE % - . $ (not to the statement @DEL without operands which resets all the values in the work file).

Deleting the copy buffer

The value C must be entered in the EAMFILE field of control block EDTAMCB.

The value must be left-aligned and padded with blanks.

The buffers EDTKEY1 and EDTKEY2 do not have to be specified when the copy buffer is deleted.

Call

The following specifications are required (see overview):

- Entry of values in the required fields in the EDTGLCB and EDTAMCB control blocks.
- Entry of values in the EDTKEY1 buffer
- Entry of values in the EDTKEY2 buffer
- Call of the entry point address IEDTDEL with the parameter list

Overview

(For the control blocks, see section “Generation and structure of the control blocks”).

Entry point address	:	IEDTDEL

Parameter list	:	A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY2)

Call parameter		Return parameter	
EDTAMCB:	EAMFILE EAMLKEY1 EAMLKEY2	EDTGLCB:	EGLRETC EGLRMSG

EDTKEY1 EDTKEY2			

For the possible return codes, see section “Logical record access functions”.

Call in the C program

Required include files:

```
#include <stdio.h>

#include <iedtgle.h>
```

The EDTAMCB control block is declared and initialized as follows:

```
iedamcb amcb = IEDAMCB_INIT;
char key1[] = "08000001";
char key2[] = "99999999";
amcb.length_key1 = amcb.length_key2 = 8;
```

If, for example, the records from line number 800.0001 through to the end of work file 1 are to be deleted:

```
strncpy(amcb.filename,"$1 ",8);
```

In the C program, the IEDTDEL function is called as follows:

```
IEDTDEL(&glcb,&tamcb,key1,key2);
```

3.5.6 IEDTREN - Modify the line number

This access function is used to modify the line number of a record in a work file.

`EDTKEY1`: Line number that is to be modified

`EDTKEY2`: New line number

If a record with the new line number already exists or if the work file contains further records between the old and new line numbers then the function is not executed.

Call

The following specifications are required (see overview):

- Entry of values in the required fields in the `EDTGLCB` and `EDTAMCB` control blocks.
- Entry of values in the `EDTKEY1` buffer
- Entry of values in the `EDTKEY2` buffer
- Call of the entry point address `IEDTREN` with the parameter list

Overview

(For the control blocks, see section “Generation and structure of the control blocks”).

Entry point address	:	`IEDTREN`

Parameter list	:	`A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY2)`

Call parameter		Return parameter	

`EDTGLCB:`	`EGLUNIT`
	`EGLVERS`
		`EDTGLCB:`	`EGLRETC`
			
			`EGLRMSG`

`EDTAMCB:`	`EAMUNIT`
	`EAMVERS`
	`EAMFILE`
	`EAMLKEY1`
	
	`EAMLKEY2`		

`EDTKEY1` `EDTKEY2`			

For the possible return codes, see section “Logical record access functions”.

Call in the C program

Required include files:

```
#include <stdio.h>
#include <iedtgle.h>
```

The EDTAMCB control block is declared and initialized as follows:

```
iedamcb amcb = IEDAMCB_INIT;
char key1[] = "01234000";
char key2[] = "00000100";
amcb.length_key1 = amcb.length_key2 = 8;
```

If, for example, the record with line number 123.4 is the 1st record in work file 1 and is to

receive line number 0.01:

```
strncpy(amcb.filename,"$1 ",8);
```

In the C program, the IEDTREN function is called as follows:

```
IEDTREN(&glcb,&amcb,key1,key2);
```

3.5.7 IEDTGET - Read the global parameter settings

The IEDTGET function can be used to read information concerning the global EDT parameter settings. For this to be possible, the value G must be entered in the EAMFILE field of control block EDTAMCB (left-aligned and padded with blanks).

If the V17 format of the control block is used, it is no longer necessary to enter the length of the control block EDTPARG in the field EAMPREC (EDTAMCB) since the length of the output is unambiguously defined by the EDTPARG version. Similarly, no values are entered for EAMLREC on return.

EDT stores the information in the EDTPARG control block (see section “EDTPARG - Global parameter settings”).

Call

The following specifications are required (see overview):

- Entry of values in the required fields in the EDTGLCB and EDTAMCB control blocks.
- Specification of the *initialized* EDTPARG control block as the output area
- Call of the entry point address IEDTGET with the parameter list

Overview

(For the control blocks, see section “Generation and structure of the control blocks”).

Entry point address	:	IEDTGET

Parameter list	:	A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY, EDTPARG)

Any values can be entered for the buffers EDTKEY1 and EDTKEY as these are not evaluated.

Call parameter		Return parameter	
EDTGLCB:	EGLUNIT EGLVERS	EDTGLCB:	EGLRETC EGLRMSG
EDTAMCB:	EAMUNIT EAMVERS EAMFILE	EDTPARG	
EDTPARG:	EPGUNIT EPGVERS		

Return codes on status query

EGLMRET	EGLSR1

EAMRETOK	EAMOK00
EAMACERR	
EAMPAERR	EAMAC12
EAMPAERR	
EAMPAERR	EAMPA04
EAMPAERR	
EAMPAERR	EAMPA08
	EAMPA12
	EAMPA32
	EAMPA36

If the call is successful (return code is EAMRETOK/EAMOK00) then the information is entered in the control block EDTPARG.

For the meaning of the return codes, see section “EDTGLCB - Global EDT control block”.

If the call is not successful, the control block EDTPARG remains unchanged.

Call in the C program

Required include files:

```
#include <stdio.h>

#include <iedtgle.h>
```

The control blocks EDTAMCB and EDTPARG are declared and initialized as follows:

```
iedamcb amcb = IEDAMCB_INIT;
iedparg parg = IEDPARG_INIT;
strncpy(amcb.filename,"G       ", 8);
```

In the C program, the IEDTGET function for reading the global parameter settings is called as follows:

```
IEDTGET(&glcb,&amcb,NULL,NULL,&parg);
```

3.5.8 IEDTGET - Read work file-specific parameter settings

The IEDTGET function can be used to read the specific parameter settings for a work file. One of the following values must be entered in the EAMFILE field of control block EDTAMCB.

L0..L22 (work file 0..22)

The values must be left-aligned and padded with blanks.
The length of the control block EDTPARL no longer has to be entered in the field EAMPREC (EDTAMCB) since the length of the output is unambiguously defined by the EDTPARL version. Similarly, no values are entered for EAMLREC on return.
EDT stores the information in the EDTPARL control block (see section “EDTPARL - Work file specific parameter settings”).

Call

The following specifications are required (see overview):

- Entry of values in the required fields in the EDTGLCB and EDTUPCB control blocks.
- Specification of the initialized EDTPARL control block
- Call of the entry point address IEDTGET with the parameter list

Overview

(For the control blocks, see section “Generation and structure of the control blocks”).

Entry point address	:	IEDTGET

Parameter list	:	A (EDTGLCB, EDTAMCB, EDTKEY1, EDTKEY, EDTPARL)

Any values can be entered for the buffers EDTKEY1 and EDTKEY as these are not evaluated.

Call parameter		Return parameter	
EDTGLCB:	EGLUNIT EGLVERS	EDTGLCB:	EGLRETC EGLRMSG
EDTAMCB:	EAMUNIT EAMVERS EAMFILE	EDTPARL	
EDTPARL:	EPLUNIT EPLVERS		

Return codes on status query

EGLMRET	**EGLSR1**

EAMRETOK	EAMOK00
EAMACERR	EAMAC12
EAMPAERR	EAMPA04
EAMPAERR	EAMPA08
EAMPAERR	EAMPA12
EAMPAERR	EAMPA32
EAMPAERR	EAMPA36

If the call is successful (return code is EAMRETOK/EAMOK00) then the information is entered in control block EDTPARL.

For the meaning of the return codes, see section “EDTGLCB - Global EDT control block”.

If the call is not successful, the control block EDTPARL remains unchanged.

Call in the C program

Required include files:

```
#include <stdio.h>

#include <iedtgle.h>
```

The EDTAMCB control block is declared and initialized as follows:

```
iedamcb amcb = IEDAMCB_INIT;
iedparl parl = IEDPARL_INIT;
strncpy(amcb.filename,"L1      ",8);
```

In the C program, the IEDTGET function for reading the work file-specific parameter settings is called as follows:

```
IEDTGET(&glcb,&amcb,NULL,NULL,&parl);
```

4 User defined statements - @USE

EDT allows users to create their own statements. To this end, the function of the statement must be implemented in the form of an external routine (statement routine). This routine is usually stored as a module in a library. However, it is also possible to integrate statement routines directly in a main program which then starts EDT via the subroutine interface.

A statement routine is defined with the @USE statement and can be called by means of the user statement symbol declared in @USE.

If a text follows the user-defined statement then this is passed to the statement routine. When control is returned, the statement routine can pass a message to EDT.

4.1 Declaring a user-defined statement

The @USE statement makes it possible to declare a user-defined statement.
A detailed presentation and description of the @USE statement can be found in the Statements User Guide [1].

@USE COMMAND = ‘spec’ ,ENTRY = entry, MODLIB = modlib

or in compatible format

@USE COMMAND = ‘spec’ (name[,modlib])

4.2 Calling a user-defined statement

Entering the defined user statement symbol (spec) for a user-defined statement starts the corresponding statement as a subroutine.

Any required text can be passed to the statement routine.

If an entry point (@USE COMMAND= ...,ENTRY=entry,...) was defined at the time of declaration with @USE, then the entire text following the user statement symbol is passed to the statement routine when the user-defined statement is entered. This text can be interpreted by the statement routine as required.

If no entry point (@USE COMMAND= ...,ENTRY=*,...) was defined at the time of declaration with @USE, then, when the user-defined statement is entered, the name of theentry point (entry) must be specified. After this, a string is specified and this is passed to the statement routine.

```
spec entry [chars]
```

In this case, entry can be considered as the statement name and chars as an operand. If the compatible format of the @USE statement has been used, then at most the first 8 characters (in uppercase and converted to the EDF03IRV character set) of the user-defined statement are considered to constitute the name of the entry point.

If the new format has been used, then at most the first 32 characters (in uppercase and converted to the EDF03IRV character set) are considered to constitute the name of the entry point.

Entry points which are specified in this way in the user-defined statement itself must therefore not contain any lowercase characters.

It is advisable to separate the statement name in this type of user-defined statement from the remainder of the statement with a space in order to avoid interpretation errors due to this truncation after 8 or 32 characters.

If it is not possible to convert the text that is to be passed to the statement routine into the character set which was defined via the initialization routine (see “Calling the initialization routine for a user-defined statement”) then the statement is rejected with an error message.

If the user statement symbol is identical to the current EDT statement symbol then the EDT statement has priority. This means that only if no EDT statement (in any permitted abbreviated form) can be identified is it assumed that the statement in question is a user statement. It is the user's responsibility to avoid conflicts.

When the statement routine is branched to, the registers are loaded as follows:

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

For an explanation of the register contents, see section “Calling EDT”.

Parameter list	:	A (EDTGLCB, COMMAND)

EDT passes the following parameters to the statement routine:

- **EDTGLCB**
  Global EDT control block which is supplied by EDT and is already initialized.
  The control block may no longer be modified once the user routine is exited.

  The version of the EDTGLCB control block that is used is determined via the initialization routine (see section "Calling the initialization routine for a user-defined statement"). EDT passes the key used to send the statement in the EGLCDS field (only in the case of statement filters, see section "Special application as statement filter").

  The control block is described in section "EDTGLCB - Global EDT control block".
- **COMMAND**
  Buffer provided by EDT which contains the string which was specified on the entry of the external statement.

  The user statement symbol is not passed. The maximum length is 32763 bytes + 4 byte length field (see section "Buffers").

  The COMMAND buffer is coded in the character set which was declared via the initialization routine (see the section on calling the initialization routine for a user-defined statement).

  EDT enters the name of the character set in the EGLCCSN field.
  The character set may be converted to uppercase before being passed to the statement routine depending on the @PAR LOW setting.

Note

The remainder of the description relates to a call issued with version 2 of the EDTGLCB control block. For a description of a call using version 1 of the control block, see [3].

A statement routine should always query the version of the received EDTGLCB control block.

Overview

EDT calls the statement routine with the following parameter list:

Parameter list	:	A (EDTGLCB, COMMAND)

Call parameter		Return parameter	
Values supplied by EDT before the external statement routine is called		Evaluated by EDT after control is returned from the external statement routine	
EDTGLCB:	EGLUNIT EGLVERS EGLCCSN EGLCDS EGLUSR1 EGLUSR2	EDTGLCB:	EGLRETC EGLRMSG EGLCMDS EGLFILE EGLUSR2
COMMAND			

Return codes for user-defined statements

The return codes are set by the statement routine and evaluated by EDT. If the statement routine has not entered any message text in the EGLRMSG field then EDT issues the following messages depending on the return code:

EGLMRET	EGLSR1	Meaning
EUPRETOK	00	No message.
	00	Message EDT3991
EUPSYERR	00	
		Message EDT5991
EUPRTERR		

If the statement routine has stored a user-defined message in the EGLRMSG field then EDT outputs *this* message instead of the messages described above (also in the case of EUPRETOK).
Depending on the return code, the statement routine's message text is used in one of the EDT messages EDT0999, EDT3999 or EDT5999.

If a different return code is received, EDT issues the message EDT5410. In this case, the user-defined message specified in the EGLRMSG field is not output.

Usually, the statement routine is intended to communicate with the calling EDT instance, i.e. it uses the EDTGLCB passed at the interface for EDT subroutine interface calls.

The statement routines can be used to execute the following functions:

- the record access functions IEDTGET, IEDTPUT, etc. (see section “Logical recordaccess functions”).
- the IEDTEXE function which executes an EDT statement (see section “IEDTEXE -Execute EDT statements without screen dialog”).

  The following statements may not be specified for IEDTEXE:

  - a further user-defined statement
  - the user routine call (@RUN)
  - the statements @DIALOG, @EDIT, @END, @HALT, @MODE and @RETURN
  - the procedure calls @DO and @INPUT.

In exceptional cases, it may be necessary for the statement routine to communicate with another EDT instance. To do this, it is necessary to use an EDTGLCB which has been initialized independently of the supplied EDTGLCB or which has not been initialized at all. Since the two EDT instances run completely independently of one another, calls to the other instance are not subject to any restrictions. If this method is used, it is not possible to access the calling instance's data.

4.3 Calling the initialization routine for a user-defined statement

If EDT is to call the statement routine of a user-defined statement using the V17
format of the interface, the developer of the statement routine must also provide an initialization routine.

The name of the initialization routine's entry point is formed from the name of the statement

routine plus the appended string `@I`.

`Example`

Entry point for the statement routine:	`SELECT`
Entry point for the associated initialization routine:	`SELECT@I`

The initialization routine must be located in the same module as the statement routine, i.e. when the statement routine is loaded, the initialization routine must also be loaded.

In the initialization routine, the user is able to define the character set in which the buffer is to be coded when the statement routine is called (in the `EGLCCSN` field in control block `EDTGLCB`). If the user leaves `EGLCCSN` empty (blanks or binary zeros), EDT assumes that the character set is `UTFE`.
In addition, the initialization routine can perform specific initialization operations for the statement routine. The initialization routine can use the functions of the `IEDTGLE` interface. The same restrictions as for statement routines apply.
The initialization routine is always called for a statement routine when the statement routine is loaded or its address is located. In this case, each entry point is considered to be a separate statement routine (even if multiple entry points designate the same location in the code).

- If no fixed entry point is specified in the @USE statement (* is specified as the entry point), then the name of the entry point is not formed until entry of the user-defined statement. The initialization routine is then called on every user statement call with this entry point. As a result, an initialization routine for this type of statement must be written in such a way that it can handle multiple calls.
- If a fixed entry point is specified in the @USE statement, then the statement routine is loaded on the @USE statement and the initialization routine is called immediately afterwards. Usually this only occurs once. However, if the user symbol has been assigned to another statement routine in the meantime then the initialization routine is also called again when the call to the first statement routine is repeated, and this irrespectively of whether or not the first statement routine had to be reloaded.

If the developer has not defined an initialization routine then the statement routine is called with version 1 of the `EDTGLCB` control block (V16 format). In this case, the `COMMAND` buffer is always coded in the `UTFE` character set.

On entry into the initialization routine, the registers are loaded as follows:

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

For an explanation of the register contents, see section “Calling EDT”.

EDT calls the initialization routine with the following parameter list:

Parameter list	:	A(EDTGLCB)

The EDTGLCB control block is supplied by EDT and is already initialized. The control block may no longer be used once the initialization routine is exited. The initialization routine is called with version 2 of the control block.

Call parameter		**Return parameter**	
Values supplied by EDT before the external initialization routine is called		Can be supplied with values by the initialization routine and are evaluated by EDT after return	
EDTGLCB:	EGLUNIT EGLVERS EGLUSR1 EGLUSR2	EDTGLCB:	EGLRETC EGLUSR2 EGLCCSN EGLINDB

Return codes for initialization routines

The return codes must be set by the initialization routine and are evaluated by EDT.

EGLMRET	**EGLSR1**	**Meaning**
EUPRETOK	00	No error
EUPVEERR	00	Version not supported by the initialization routine; the userdefined statement is rejected with the message EDT5470.
EUPRTERR	00	Other errors. The user-defined statement is rejected with the message EDT5471.

Following the return from the initialization routine, EDT evaluates the EGLCCSN field. Each time the associated statement routine is called, the COMMAND buffer is coded in this character set. The character set declared in the initialization routine simply defines the character set with which the buffer is passed to the statement routine. When EDT interfaces are called by the statement routine, the caller can of course specify another character set by overwriting the EGLCCSN field. However, this operation has no impact on subsequent statement routine calls but only applies locally for the called IEDTGLE interface.

The statement routine calls are performed using version 2 of the EDTGLCB control block.

The EGLCOMP flag in the EGLINDB indicator byte is also evaluated after the return from the initialization routine. If the initialization routine has set this field then a statement filter (see section “Special application as statement filter”) always receives the statement to be filtered in uppercase characters. If the field is not set then the statement is passed in uppercase or upper/lowercase depending on the @PAR LOW setting.

If EDT V16.6B or EDT V17.0A is running in compatibility mode then the initialization routine is not called. The statement routine call is performed using version 1 of the EDTGLCB control block.

A statement routine that is intended to run in both Unicode mode and in compatibility mode or with EDT V16.6B must query the version (EGLVERS) of the control block passed by EDT and react accordingly.

4.4 Special application as statement filter

When EDT is called via the function IEDTCMD or IEDTEXE (see section "IEDTCMD -Execute EDT statements"), the calling program is able to declare a user-defined statement with an empty user statement symbol as the statement filter:

@USE COMMAND = ' ' ,ENTRY=entry,MODLIB=modlib

or in compatible format

@USE COMMAND=' ' (name [,modlib])

Every statement which has been entered

- in F mode in the statement line
- in L mode or
- in EDT procedures

is then sent to this routine.

The routine is not sent

- statement codes entered in F mode
- statements entered via the program interfaces IEDTCMD or IEDTEXE.

The routine is also sent statements with an unknown statement name or which contain syntax errors.
If multiple statements are entered (separated by semicolons) then the individual statements are passed to the routine one after the other.
The statement symbol itself is only passed if the statement is a user statement. Leading blanks in front of the statement symbol are not passed.

EDT passes the key used to send the statement in the EGLCDS field (EDTGLCB).

If it is not possible to convert a statement into the character set which was defined for the statement filter in the initialization routine (see section "Calling the initialization routine for auser-defined statement") then the statement is rejected with an error message. Consequently, statement filters should work with a Unicode character set as far as possible (UTFE is recommended).

The initialization routines for statement filters can also specify whether the filters expect to receive statements in uppercase or uppercase/lowercase notation. In the former case, the initialization routine must set the EGLCOMP flag (EDTGLCB) in EGLINDB (see also section "Calling the initialization routine for a user-defined statement").

A statement filter can use the functions of the IEDTGLE interface. The same restrictions as for statement routines apply.

The statement passed by EDT cannot be modified in an application filter (i.e. all the changes are ignored).
However, on return to EDT, the statement routine can pass the following values in the return code EGLSR1:.

EGLMRET	EGLSR1	Meaning
EUPRETOK	EUPOK00	The statement should be executed.

EUPRET0K	EUP0K12	EDT should terminate the current statement dialog. Control is returned to the main program in the same way as if @HALT had been entered.
EUPRET0K	EUP0K24	The statement should not be executed.

Note

In Unicode mode, EDT V17.0A supports the implementation of statement and user routines in C (including correct supply to the C runtime system).
However, it should be noted that this type of routine cannot run with EDT V16.6B since this version does not call statement routines with a save area which is correctly prepared for the runtime system. This shortcoming has been eliminated in EDT V17.0A compatibility mode. However, input to the runtime system in compatibility mode is only possible if the statement routine is loaded dynamically from a library. Consequently, if a statement routine written in C is to run both in Unicode and in compatibility mode it should not be statically linked to a main program.

5 User routines - @RUN

The @RUN statement makes it possible to declare a user routine as a subroutine. A description of the @RUN statement can be found in the Statements User Guide [1].

@RUN ENTRY = ... [,MODLIB =...] [,UNLOAD] [, '...']

In Unicode mode, the user routine is called via the same interface as statement routines in a user-defined statement (see section “User defined statements - @USE”). The string specified in the statement is passed in the `COMMAND` buffer.

The only difference to a statement routine is that EDT error recovery is deactivated when a user routine is called whereas it remains active in the case of statement routines (provided that it was already active).

The user routine can use the functions of the `IEDTGLE` interface. The same restrictions as for statement routines apply.

The associated initialization routine is also called for user routines (on each call) (see the section on calling initialization routines for user-defined statements). If no initialization routine is defined then the @RUN statement is rejected with the message `EDT5469`.

Caution

> The format of the statement and interface used to call the routine are different in Unicode and compatibility mode.

6 Producing subroutine interface applications

On the basis of rules and examples, this section explains the production of programs in BS2000 which use EDT as a subroutine or which are to be called by EDT as user routines by means of the @USE or @RUN statements.

The languages C and Assembler are considered. For other languages, EDT provides only ILCS-compliant linkage. In such cases, it is the responsibility of users themselves to ensure the dynamic loading, linking and initialization of the required runtime routines.

6.1 Producing main programs in C

It is advisable to link the main program with STDLIB=*DYNAMIC (default value) and call EDT with the ILCS flag indicator.ilcs_environment = 1 (default value). The procedure below compiles and links C main programs which are stored as BEISPIEL1.C, BEISPIEL2.C, ... in the library EDT.BEISPIELE.
The generated program, the compiler lists and the diagnostic output are also stored in this library.

```
/SET-PROCEDURE-OPTIONS INPUT-FORMAT=FREE-RECORD-LENGTH,/
    DATA-ESCAPE-CHAR=STD
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME=NR
/   DECLARE-PARAMETER NAME=LIB,INITIAL-VALUE=C'EDT.BEISPIELE
/END-PARAMETER-DECLARATION
/START-CPLUS-COMPILER
//MODIFY-SOURCE-PROP LANGUAGE=*C
//MODIFY-RUNTIME-PROPERTIES PARAMETER-PROMPTING=*NO
//MODIFY-INCLUDE-LIBRARIES USER-INCLUDE-LIBRARY=//
    $.SYSLIB.EDT.170                  ------------------------------ (1)
//MODIFY-DIAGNOSTIC-PROPERTIES OUTPUT=*LIBRARY-ELEMENT(//
    LIB=&LIB,ELEM=*STD-ELEMENT()),//
    MIN-MSG-WEIGHT=*NOTE
//MODIFY-LISTING-PROPERTIES OUTPUT=*LIBRARY-ELEMENT(//
    LIB=&LIB,ELEM=*STD-ELEMENT()),SOURCE=*YES()
//COMPILE SOURCE=*LIB-ELEM(LIB=&LIB,ELEM=BEISPIELE&NR..C),//
    MODULE-OUTPUT=*LIB-ELEM(LIB=&LIB,ELEM=BEISPIEL&NR)
//MOD-BIND-PROP INCLUDE=(*LIB(LIB=&LIB,ELEM=BEISPIEL&NR),-   ---------- (2)
```

```
//    *LIB(LIB=$.SYSLNK.EDT.170,ELEM=IEDTGLE)),//
    RUNTIME-LANG=*C,STDLIB=*DYNAMIC --------------------------------- (3)
//BIND OUTPUT=*LIB(LIB=&LIB,ELEM=BSP&NR.C)  --------------------------- (4)
//END
/EXIT-PROCEDURE
```

Explanations

(1) It is assumed that SYSLIB.EDT.170 is installed under the default user ID. The C compiler needs this library in order to locate the IEDTGLE interface header files.

(2) The user program is linked to the module IEDTGLE from SYSLNK.EDT.170. IEDTGLE uses IMON to locate the EDT libraries and either loads EDT dynamically or connects to the EDT subsystem.

(3) The adapter for the C runtime system is automatically linked by the compiler if STDLIB=*DYNAMIC is specified.

(4) The generated program can then be started with /START-EXECUTABLE-PROGRAM (E=BSPxC,L=EDT.BEISPIELE) (x = 1,2,..).

The ILCS flag is only significant if EDT is to load and call user routines (see the following sections).

6.2 Producing user routines in C

A C user routine which is produced from a single source can be stored as an LLM in a library without the need for a link stage. When loading the user routine, EDT specifies a "resolve context" which contains the required runtime modules.

The precise mechanism depends on whether or not EDT was initialized by a C main program with an ILCS flag:
If this is the case then user routines are loaded in a separate load context EDT#USER and the two load contexts EDT#CRTS and LOCAL#DEFAULT (in this sequence) are specified as the resolve context. As a result, external references to the C globals in the user routine are first resolved against the entries in EDT#CRTS (EDT runtime environment) and then against the entries in LOCAL#DEFAULT (C main program).
Since a (dynamically loadable) user routine is always called in the EDT runtime environment, the correct C globals are made available to the user routine irrespectively of whether the C main program itself possesses visible entries for the C globals.
The user routine's own load context must be taken into account if additional modules are to be dynamically loaded from within the user routine using BIND.

If the C main program has initialized EDT without an ILCS flag then, for reasons of compatibility, the user routine is loaded into the load context LOCAL#DEFAULT and EDT#CRTS is specified as the resolve context. In this case, the C main program should not have any visible entries to the C globals as otherwise these override the corresponding entries in the EDT runtime environment and the user routine is not able to work correctly with the C library functions.

The procedure below compiles and links C user routines which are stored as ANWEND1.C, ANWEND2.C, ... in the library EDT.BEISPIELE.
The generated LLM, the compiler lists and the diagnostic output are also stored in this library.

```
/SET-PROCEDURE-OPTIONS INPUT-FORMAT=FREE-RECORD-LENGTH,/
    DATA-ESCAPE-CHAR=STD
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME=NR
/   DECLARE-PARAMETER NAME=LIB,INITIAL-VALUE=C'EDT.BEISPIELE
/END-PARAMETER-DECLARATION
/START-CPLUS-COMPILER
//MODIFY-SOURCE-PROP LANGUAGE=*C
//MODIFY-RUNTIME-PROPERTIES PARAMETER-PROMPTING=*NO
//MODIFY-INCLUDE-LIBRARIES USER-INCLUDE-LIBRARY=//
    $.SYSLIB.EDT.170                  ------------------------------ (1)
//MODIFY-DIAGNOSTIC-PROPERTIES OUTPUT=*LIBRARY-ELEMENT(//
    LIB=&LIB,ELEM=*STD-ELEMENT()),//
    MIN-MSG-WEIGHT=*NOTE
//MODIFY-MODULE-PROPERTIES LOWER-CASE-NAMES=*YES,-
```

```
//    SPECIAL-CHARACTERS=*KEEP          ------------------------------ (2)
//MODIFY-LISTING-PROPERTIES OUTPUT=*LIBRARY-ELEMENT(//
    LIB=&LIB,ELEM=*STD-ELEMENT()),SOURCE=*YES()
//COMPILE SOURCE=*LIB-ELEM(LIB=&LIB,ELEM=ANWEND&NR..C),-
//    MODULE-OUTPUT=*LIB-ELEM(LIB=&LIB,ELEM=ANWEND&NR)
//END
/EXIT-PROCEDURE
```

Explanations

(1) It is assumed that `SYSLIB.EDT.170` is installed under the default user ID. The C compiler needs this library in order to locate the `IEDTGLE` interface header files.

(2) It should be possible to address the entries in the user routine under their original names. Consequently, lowercase characters must be accepted and special characters (such as "_") should not be changed.

6.3 C main programs and user routines in the same program

The production of a main program which also makes user routines available is identical to the procedure described in section “Producing main programs in C”. The procedure specified there can be applied unchanged. However, the statement

```
//MODIFY-MODULE-PROPERTIES LOWER-CASE-NAMES=*YES,-
// SPECIAL-CHARACTERS=*KEEP
```

may be added to the procedure as described in section “Producing user routines in C” if the user routines are to be addressed under their original names (casesensitive).

If a C main program and user routines are present in the same program then it is *essential* for EDT to be initialized with the ILCS flag.
Before calling the user routine, EDT then switches to the C main program's
runtime system and calls the user routine in this. This switchover is not performed if the ILCS flag is not set, with the result that the user routine does not use the correct C globals and cannot work correctly with C library functions.

6.4 Producing main programs in Assembler

It is advisable to link the IEDTGLE module explicitly to the main program. For this reason, a link editor run follows assembly in the procedure below. Usually, a simple Assembler program calls EDT without an ILCS flag (EGLILCS). This is the default behavior when generating the IEDTGLCB control block. The procedure below assembles and links Assembler main programs which are stored as BEISPIEL1.ASS, BEISPIEL2.ASS, ... in the library EDT.BEISPIELE.
The generated program and the compiler lists are also stored in this library.

```
/SET-PROCEDURE-OPTIONS INPUT-FORMAT=FREE-RECORD-LENGTH,/
    DATA-ESCAPE-CHAR=STD
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME=NR
/   DECLARE-PARAMETER NAME=LIB,INITIAL-VALUE=C'EDT.BEISPIELE
/END-PARAMETER-DECLARATION
/START-ASSEMBH
//COMPILE SOURCE=*LIBRARY-ELEMENT(//
    LIBRARY=&LIB,ELEMENT=BEISPIEL&NR..ASS),//
    MACRO-LIBRARY=($.SYSLIB.EDT.170,$.MACROLIB),-        ------------ (1)
//    COMPILER-ACTION=*MODULE-GENERATION(MODULE-FORMAT=*LLM),//
    MODULE-LIBRARY=&LIB,//
    LISTING=*PARAMETERS(OUTPUT=*LIBRARY-ELEMENT(//
        LIBRARY=&LIB,ELEMENT=BEISPIEL&NR..LST))
//END
/START-BINDER
//START-LLM-CREATION INTERNAL-NAME=BEISPIEL&NR._ASS
//INCLUDE-MODULES MODULE-CONTAINER=*LIBRARY-ELEMENT(//
    LIBRARY=&LIB,ELEMENT=BEISP&NR)                       ------------ (2)
//INCLUDE-MODULES MODULE-CONTAINER=*LIBRARY-ELEMENT(//
    LIBRARY=$.SYSLNK.EDT.170,ELEMENT=IEDTGLE)            ------------ (3)
//SAVE-LLM MODULE-CONTAINER=*LIBRARY-ELEMENT(//
    LIBRARY=&LIB,ELEMENT=BSP&NR.A)                       ------------ (4)
//END
/EXIT-PROCEDURE
```

Explanations

(1) It is assumed that SYSLIB.EDT.170 is installed under the default user ID. The Assembler needs this library in order to locate the IEDTGLE interface macros. It is not usually necessary to specify the default macro library but it may be necessary to specify further macro libraries.

(2) It is assumed that the name of the first CSECT in the Assembler program is BEISPx (BEISPIELx would be too long in standard Assembler).

(3) The user program is linked to the module IEDTGLE from SYSLNK.EDT.170. IEDTGLE uses IMON to locate the EDT libraries and either loads EDT dynamically or connects to the EDT subsystem.

(4) The generated program can then be started with /START-EXECUTABLE-PROGRAM (E=BSPxA,L=EDT.BEISPIELE) (x = 1,2,..).

6.5 Producing user routines in Assembler

It is advisable *not* to link the IEDTGLE module. Otherwise it would be necessary to hide the entries in this module with the BINDER link editor as otherwise "duplicate entries" would occur when multiple user routines are loaded.
On dynamic loading, the module's open externals are then either resolved against an IEDTGLE which has already been loaded with the main program or another user routine or the IEDTGLE which is preloaded with the EDTCON subsystem is used.
The procedure below assembles user routines which are stored as ANWEND1.ASS, ANWEND2.ASS, ... in the library EDT.BEISPIELE.
The generated program and the compiler lists are also stored in this library.

```
/SET-PROCEDURE-OPTIONS INPUT-FORMAT=FREE-RECORD-LENGTH,/
    DATA-ESCAPE-CHAR=STD
/BEGIN-PARAMETER-DECLARATION
/   DECLARE-PARAMETER NAME=NR
/   DECLARE-PARAMETER NAME=LIB,INITIAL-VALUE=C'EDT.BEISPIELE
/END-PARAMETER-DECLARATION
/START-ASSEMBH
//COMPILE SOURCE=*LIBRARY-ELEMENT(//
    LIBRARY=&LIB,ELEMENT=ANWEND&NR..ASS),//
    MACRO-LIBRARY=($.SYSLIB.EDT.170,$.MACROLIB),-            -------- (1)
//    COMPILER-ACTION=*MODULE-GENERATION(MODULE-FORMAT=*LLM),//
    MODULE-LIBRARY=&LIB,//
    LISTING=*PARAMETERS(OUTPUT=*LIBRARY-ELEMENT(//
        LIBRARY=&LIB,ELEMENT=ANWEND&NR..LST))
//END
/EXIT-PROCEDURE
```

Explanations

(1) It is assumed that SYSLIB.EDT.170 is installed under the default user ID. The Assembler needs this library in order to locate the IEDTGLE interface macros.
It is not usually necessary to specify the default macro library. It may be necessary to specify further macro libraries.

7 Examples

7.1 Example 1 - C main program

The following example program uses only the `IEDTCMD` interface.

```
/*********************************************************************/
/*                                                                   */
/* Example 1                                                         */
/*                                                                   */
/* This example uses only the IEDTCMD                                */
/* interface to execute EDT statements.                              */
/*                                                                   */
/* The example program performs the following actions:               */
/*                                                                   */
/* 1) Read a selection criterion (CCSN)                              */
/* 2) Output a table of contents to work file 0                      */
/*    using the @SHOW statement (Format 1).                          */
/* 3) Delete all the lines which do not contain the search           */
/*    criterion with the @ON statement (Format 10).                  */
/* 4) Set work file 0 and then switch to the F mode                  */
/*    dialog by means of a @SETF statement followed by a             */
/*    @DIALOG statement.                                             */
/* 5) The user can now edit the lines that are output                */
/*    and after that terminate EDT and consequently also this       */
/*    example program.                                               */
/*                                                                   */
/*********************************************************************/
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
/* Include files of the EDT subroutine interface */
#define EDT_V17                         /*--------------------------  (1) */
#include <iedtgle.h>                    /*--------------------------- (2) */
/* Create and initialize the EDT subroutine interface data          */
/* structures required for this example.                            */
```

```
static iedglcb glcb = IEDGLCB_INIT;
static iedupcb upcb = IEDUPCB_INIT;
static iedbuff *command = NULL;
static iedbuff *message1 = NULL;
static iedbuff *message2 = NULL;
/*********************************************************************/
/* Function: printrc                                                 */
/*                                                                   */
/* Task:                                                             */
/* If EDT has returned an error message, then the error message      */
/* is output by this function.                                       */
/*                                                                   */
/* Parameter: errmsg   (IN)  Pointer to the additional error         */
/*                           message to be output if an error occurs*/
/*                                                                   */
/* Return value: none                                                */
/*********************************************************************/
static void
printrc(char *errmsg)
{
    char message[81];
    if ((glcb.rc.structured_rc.mc.maincode != 0) &&
         (glcb.return_message.structured_msg.rmsgl > 0))
    {
        printf("%s\n",errmsg);   /* Output the passed error message */
        strncpy(message,(char*)glcb.return_message.structured_msg.rmsgf,
            glcb.return_message.structured_msg.rmsgl);
        message[glcb.return_message.structured_msg.rmsgl] = 0x00;
        printf("Meldungstext: %s\n",message);  /* Output EDT message */
        exit(1);
    }
}
/*********************************************************************/
/* Function: fill_buff                                               */
/*                                                                   */
/* Task:                                                             */
/* This function enters content and the record length field in a     */
/* record of variable length.                                        */
/*                                                                   */
/* Parameter: p:       (IN)  Pointer to a structure of type          */
/*                           iedbuff, which contains the variable    */
/*                           length record to be set.                */
/*            textp:   (IN)  Points to a string which contains the   */
/*                           text to be entered. The length          */
/*                           of the string implicitly defines the    */
```

```
/*                          record length (length of string + 4).  */
/*                                                                 */
/* Return value: none                                              */
/*******************************************************************/
static void
fill_buff(iedbuff *p,char *textp)
{
    size_t l_text;                          /* Length of string */
    if ((l_text = strlen(textp)) > 2044)                /*---------- (3) */
        l_text = 2044;     /* Restrict length to 2044 characters   */
    strncpy((char *)p->text,textp,l_text);  /* Enter text          */
    p->length = l_text + 4;                 /* Enter record length */
}
/*******************************************************************/
/* Function: edtcmd                                                */
/*                                                                 */
/* Task:                                                           */
/* This function enters the passed strings in records of           */
/* variable lengths (DMS format) and then calls the EDT's          */
/* CMD interface.                                                  */
/*                                                                 */
/* Parameter: cmd:     (IN)  Pointer to a string which contains    */
/*                           the EDT statement(s) that are to be   */
/*                           executed. The length of the string    */
/*                           implicitly defines the record length  */
/*                           (String length + 4).                  */
/*            msg1:    (IN)  Pointer to a string which contains the */
/*                           text to be entered. The length        */
/*                           of the string implicitly defines the  */
/*                           record length (length of string + 4).  */
/*            msg2:    (IN)  Pointer to a string which contains the */
/*                           text to be entered. The length        */
/*                           of the string implicitly defines the  */
/*                           record length (length of string + 4).  */
/*                                                                 */
/* Return value: none                                              */
/*******************************************************************/
static void
edtcmd(char *cmd,char *msg1,char *msg2)
{
    fill_buff(command,cmd);
    fill_buff(message1,msg1);
    fill_buff(message2,msg2);
    IEDTCMD(&glcb,&upcb,command,message1,message2);
}
```

```
/*****************************************************************/
/* Main program                                                  */
/*****************************************************************/
int
main(void)
{
    char input[81];     /* Input area */
    char ccsn[9];       /* Entered CCSN */
    char cmd[257];      /* Area for the construction of EDT statements */
    /* Provide buffer */
    command = (iedbuff *)malloc(2048);
    message1 = (iedbuff *)malloc(2048);
    message2 = (iedbuff *)malloc(2048);
    printf("\nStart Beispiel1\n\n");
    /* Read in CCSN */
    printf("Bitte CCSN eingeben (UTFE,UTF16,EDFxxx): ");
    scanf("%s",input);
    if (strlen(input) < 1 || strlen(input) > 8)
    {
         printf("Eingabe zu kurz oder zu lang!");
         exit(1);
    }
    strupper(ccsn,input);      /* Conv. CCSN into uppercase */
    /*  Output table of contents to work file 0 */
    edtcmd("SHOW F=* TO 1 LONG","","");
    printrc("Fehler bei der @SHOW-Anweisung!");
    /*  Delete all the lines which do not correspond to the search */
    /*  criterion (CCSN) and renumber the remaining lines          */
    sprintf(cmd,"ON &:100-107: FIND NOT '%s' DELETE;RENUMBER",ccsn);
    edtcmd(cmd,"","");
    printrc("Fehler bei der @ON- oder der @RENUMBER-Anweisung!");
    /*  Go to work file 0 and switch to the            */
    /*  F mode dialog                                  */
    edtcmd("SETF(0);DIALOG","Beispiel 1 fuer die UP-Schnittstelle","");
    printrc("Fehler bei der @SETF- oder der @DIALOG-Anweisung!");
    edtcmd("HALT","","");
    printf("\nEnde Beispiel1\n\n");
    return 0;
}
```

Explanations

(1) #define specifies that the V17 variant of the interface is to be generated. This causes EDT to be automatically started in Unicode mode.

(2) In EDT V17.0, all that is required is an #include statement for iedtgle.h. This header file causes the inclusion of the other required headers.

(3) In this program, the malloc for the buffer areas causes the restriction to 2044. EDT itself would accept 32767 bytes.

If the procedure explained in the section on producing main programs in C is stored in a file named `CC.DO` in BS2000 and the source file is stored as the S element `BEISPIEL1.C` in the library `EDT.BEISPIELE` then the above program can be compiled and linked with

```
/CALL-PROC CC.DO,(1).
```

The generated program can then be executed using

```
/START-EXECUTABLE-PROGRAM (E=BSP1C,L=EDT.BEISPIELE)
```

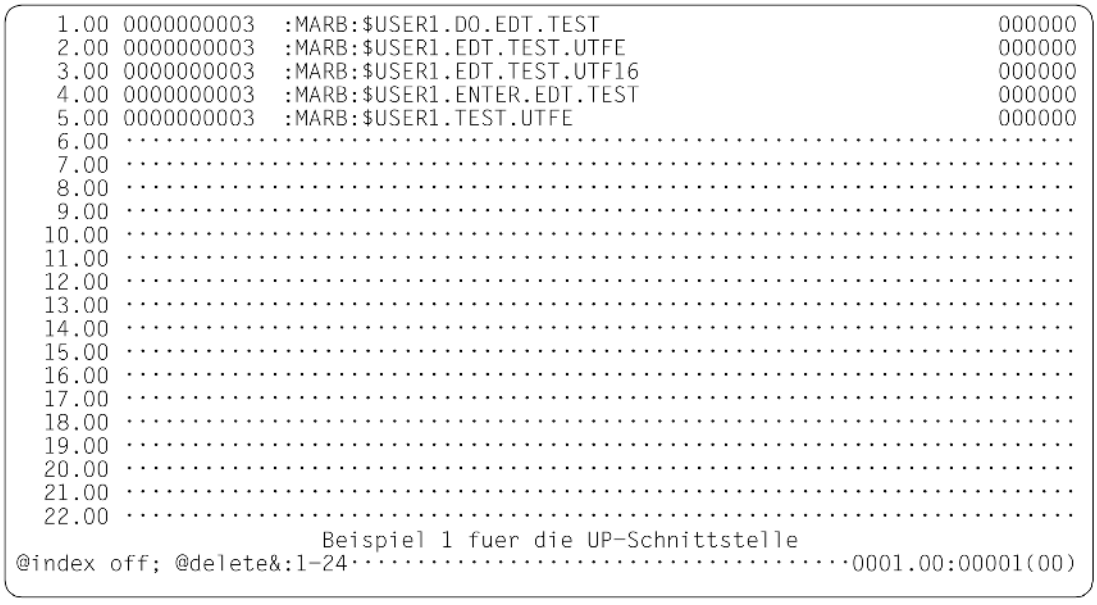
When `CC.DO` runs, the following or similar output is generated by the system or the compiler:

```
%  BLS0523 ELEMENT 'SDFCC', VERSION '031', TYPE 'L' FROM LIBRARY
':MARS:$TSOS.SYSLNK.CPP-RS.031' IN PROCESS
%  BLS0524 LLM 'SDFCC', VERSION '03.1A40' OF '2005-02-03 16:16:36' LOADED
%  BLS0551 COPYRIGHT (C) Fujitsu Siemens Computers GmbH 2005. ALL RIGHTS
RESERVED
%  CDR9992 : BEGIN C/C++(BS2000/OSD) VERSION 03.1A40
%  CDR9907 : NOTES: 0  WARNINGS: 0  ERRORS: 0  FATALS: 0
%  CDR9937 : MODULES GENERATED, CPU TIME USED = 2.4800 SEC
%  BND3102 SOME WEAK EXTERNS UNRESOLVED
%  BND1501 LLM FORMAT: '1'
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'UNRESOLVED EXTERNAL'
%  CDR9936 : END; SUMMARY: NOTES: 0  WARNINGS: 0  ERRORS: 0  FATALS: 0
%  CCM0998 CPU TIME USED: 3.4223 SECONDS
```

When `/START-EXECUTABLE-PROGRAM (E=BSP1C,L=EDT.BEISPIELE)` is called, the following or similar messages are displayed:

```
%  BLS0523 ELEMENT 'BSP1C', VERSION '@', TYPE 'L' FROM LIBRARY
':A:$USER.EDT.BEISPIELE' IN PROCESS
%  BLS0524 LLM '$LIB-ELEM$EDT$BEISPIELE$$BSP1C$$',VERSION' 'OF'2007-05-03 14
:42:12' LOADED
Start Beispiel1
Bitte CCSN eingeben (UTFE, UTF16, EDFxxx):
```

If `UTF` is entered here, i.e. if all the files coded in a Unicode character set are selected, the program next switches to the EDT dialog and the following screen is displayed:



Entering @INDEX OFF; @DELETE&:1-24 makes it possible to restrict the output to the relevant information and displays the following screen:

```
DO.EDT.TEST                                0000000001 2006-10-05 NW SAM  NN UTFE
EDT.TEST.UTFE                              0000000001 2007-04-23 NW SAM  NN UTFE
EDT.TEST.UTF16                             0000000001 2007-04-23 NW SAM  NN UTF16
ENTER.EDT.TEST                             0000000001 2007-10-05 NW SAM  NN UTFE
TEST.UTFE                                  0000000001 2007-03-22 NW SAM  NN UTF16
................................................................................
................................................................................









.....................................................................0001.00:00001(00)
```

7.2 Example 2 - C main program

This example uses the IEDTCMD interface and IEDTGET-interface to read and process the records in a file:

```
/*****************************************************************/
/*                                                               */
/* Example 2                                                     */
/*                                                               */
/* This example uses the IEDTCMD interface                       */
/* to execute EDT statements and the IEDGET                      */
/* interface to read lines. All the records in a                 */
/* file which contains accounting data are output                */
/* if they have an invoice date more than a                      */
/* predefined number of days before the current date.            */
/*                                                               */
/*                                                               */
/* The example program performs the following individual         */
/* actions:                                                      */
/*                                                               */
/* 1) Determine the current date.                                */
/* 2) Use the @COPY statement (Format 1) to read all the lines of   */
/*    the file together with their invoice dates into work area $0. */
/* 3) Execute a loop in which all the lines in work area         */
/*    $0 are read one after the other and calculate the time     */
/*    difference between the current date and the invoice date.  */
/*    If this time difference is greater than the predefined     */
/*    limit, then the customer number, invoice number and invoiced  */
/*    amount specified in this line are output.                  */
/* 4) Once this loop has terminated, EDT is terminated with the  */
/*    @HALT statement and the example program is then exited.    */
/*                                                               */
/*****************************************************************/
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
/* Include files of the EDT subroutine interface */
#define EDT_V17
#include <iedtgle.h>
/* Create and initialize the EDT subroutine interface data  */
/* structures required for this example.                    */
```

```
static iedglcb glcb = IEDGLCB_INIT;
static iedupcb upcb = IEDUPCB_INIT;
static iedamcb amcb = IEDAMCB_INIT;
static iedbuff *command = NULL;
static iedbuff *message1 = NULL;
static iedbuff *message2 = NULL;
/* Definition of an output line */
typedef struct line
{
    char kdnr[8];         /* Customer number */
    char free1[2];
    char renr[8];         /* Invoice number  */
    char free2[2];
    char betr[10];        /* Invoice amount  */
    char free3[2];
    char day[2];          /* Day of invoicing   */
    char mon[2];          /* Month of invoicing */
    char trenn2[1];
    char year[4];         /* Year of invoicing  */
    char rest[215];
} LINE;
/*********************************************************************/
/* Function: printrc                                                 */
/*                                                                   */
/* Task:                                                             */
/* If EDT has returned an error message, then the error message      */
/* is output by this function.                                       */
/*                                                                   */
/* Parameter: errmsg   (IN)  Pointer to the additional error         */
/*                           message to be output if an error occurs*/
/*                                                                   */
/* Return value: none                                                */
/*********************************************************************/
static void
printrc(char *errmsg)
{
    char message[81];
  if (glcb.IEDGLCB_RC_MAINCODE != 0)
    {
        printf("%s: %08x\n",errmsg,
         glcb.IEDGLCB_RC_NBR);/* Output passed error message */
        if (glcb.return_message.structured_msg.rmsgl > 0)
        {
            strncpy(message,(char*)glcb.return_message.structured_msg.rmsgf,
```

```
               glcb.return_message.structured_msg.rmsgl);
            message[glcb.return_message.structured_msg.rmsgl] = 0x00;
            printf("Meldungstext: %s\n",message);  /* Output EDT message */
        }
        exit(1);
    }
}
/*********************************************************************/
/* Function: fill_buff                                               */
/*                                                                   */
/* Task:                                                             */
/* This function enters content and the record length field in a     */
/* record of variable length.                                        */
/*                                                                   */
/* Parameter: p:       (IN)  Pointer to a structure of type          */
/*                           iedbuff, which contains the variable    */
/*                           length record to be set.                */
/*            textp:   (IN)  Points to a string which contains the   */
/*                           text to be entered. The length          */
/*                           of the string implicitly defines the    */
/*                           record length (length of string + 4)    */
/*                                                                   */
/* Return value: none                                                */
/*********************************************************************/
static void
fill_buff(iedbuff *p,char *textp)
{
    size_t l_text;                          /* Length of string */
    if ((l_text = strlen(textp)) > 2044)
        l_text = 2044;     /* Restrict length to 2044 characters   */
    strncpy((char *)p->text,textp,l_text);  /* Enter text          */
    p->length = l_text + 4;                 /* Enter record length */
}
/*********************************************************************/
/* Function: edtcmd                                                  */
/*                                                                   */
/* Task:                                                             */
/* This function enters the passed strings in records of             */
/* variable lengths (DMS format) and then calls the EDT's            */
/* CMD interface.                                                    */
/*                                                                   */
/* Parameter: cmd:     (IN)  Pointer to a string which contains      */
/*                           the EDT statement(s) that are to be     */
/*                           executed. The length of the string      */
/*                           implicitly defines the record length    */
```

```
/*                           (String length + 4).                   */
/*            msg1:    (IN)  Pointer to a string which contains the */
/*                           text to be entered. The length         */
/*                           of the string implicitly defines the   */
/*                           record length (length of string + 4).  */
/*            msg2:    (IN)  Pointer to a string which contains the */
/*                           text to be entered. The length         */
/*                           of the string implicitly defines the   */
/*                           record length (length of string + 4).  */
/*                                                                  */
/* Return value: none                                               */
/********************************************************************/
static void
edtcmd(char *cmd,char *msg1,char *msg2)
{
    fill_buff(command,cmd);
    fill_buff(message1,msg1);
    fill_buff(message2,msg2);
    IEDTCMD(&glcb,&upcb,command,message1,message2);
}
/********************************************************************/
/* Function: edtget                                                 */
/*                                                                  */
/* Task:                                                            */
/* This function uses the subroutine interface's GET function       */
/* to read a line from work file $0. The read operation is always   */
/* performed relative to the record with line number 0.             */
/*                                                                  */
/* Parameter: rec:     (IN)  Pointer to a data area in which        */
/*                           the line read by the GET function      */
/*                           is stored.                             */
/*            disp:    (IN)  Specifies the displacement of the      */
/*                           line to be read from line number 0.    */
/* Return value: none                                               */
/********************************************************************/
static void
edtget(char *rec,int disp)
{
    char localfile[9] = "$0      ";
    iedbyte key1[8] = {0,0,0,0,0,0,0,0};
    iedbyte key[8] = {0,0,0,0,0,0,0,0};
    /* Enter values in IEDAMCB control block */
    IEDAMCB_SET_NO_MARKS(amcb);
    amcb.length_key_outbuffer = 8;
    amcb.length_rec_outbuffer = 256;
    amcb.length_key1 = 8;              /* ------------------------- (1) */
```

```
    amcb.displacement = disp;  /* read <disp> records after 0 */
    strncpy((char *)amcb.filename,localfile,8); /* Work file $0 */
    IEDTGET(&glcb,&amcb,key1,key,rec);
}
/*****************************************************************/
/* Main program                                                  */
/*****************************************************************/
int
main(void)
{
    char filename[] = "edt.rechnung2";
    char cmd[257];
    LINE zeile;
    int  disp;
    int  days = 20;
    int  time_diff;
    time_t zeit1;
    time_t zeit2;
    struct tm t;
    printf("\nStart Beispiel2\n\n");
    /* Provide buffer */
    command = (iedbuff *)malloc(2048);
    message1 = (iedbuff *)malloc(2048);
    message2 = (iedbuff *)malloc(2048);
    /* Determine current date */
    zeit1 = time((time_t *) 0);
    /* For the conversion of the date in the line */
    /* the time is set to 00.00                   */
    t.tm_sec = 0;
    t.tm_min = 0;
    t.tm_hour = 0;
    /* Read the file to be processed using   */
    /* the @COPY statement (format 1)        */
    sprintf(cmd,"COPY FILE=%s",filename);
    edtcmd(cmd,"","");
    printrc("Fehler bei der @COPY-Anweisung!");
    /* The following loop processes all the lines */
    /* in work file $0.                           */
    for (disp = 1; disp < 99999999; disp++)   /*-------------------- (2) */
```

```
    {
        /* Read next line */
        edtget((char *)&zeile,disp);
        printrc("Fehler bei der Funktion GET!");
        /* If reading goes beyond the last line number then */
        /* IEDTGET returns "last record"                    */
        if (glcb.IEDGLCB_RC_SUBCODE1 == IEDGLCBlast_record)
            break; /* Exit loop */
        t.tm_mday = atoi(zeile.day);          /* Fetch date from  */
        t.tm_mon = atoi(zeile.mon) - 1;       /* the read-in      */
        t.tm_year = atoi(zeile.year) - 1900;  /* line             */
        zeit2 = mktime(&t);                   /* Conv. date to    */
                                              /* time value       */
        /* Determine time difference in days */
        time_diff = difftime(zeit1,zeit2)/86400;
        /* If the predefined period has expired, output the    */
        /* customer number, invoice number and invoiced amount */
        /* for the current line.                               */
        if (time_diff > days)
        {
            zeile.free1[0] = '\0';   /* --------------------------- (3) */
            zeile.free2[0] = '\0';
            zeile.free3[0] = '\0';
            printf("Kdnr.: %s, Rechn.nr.: %s, Betrag: %s Euro\n",
                 zeile.kdnr,zeile.renr,zeile.betr);
        }
    }
    /* Terminate EDT and program */
    edtcmd("HALT","","");
    printrc("Fehler bei der @HALT-Anweisung!");
    printf("\n\nEnde Beispiel2\n\n");
    return 0;
}
```

Explanations

(1) In the `IEDTGET` function it is only necessary to enter values for `length_key1` and `length_key_outbuffer.`

(2) The method for the sequential reading of the work file implemented here is not particularly effective since it is always necessary to restart the read operation from the beginning again. Readers should consider how it might be possible to develop a more effective algorithm by using the key returned in `key` with a constant `amcb.displacement = +1.`

(3) The end character for C strings is used here to permit direct output without conversion.

If the procedure explained in the section “Producing main programs in C” is stored in a file named `CC.DO` in BS2000 and the source file is stored as the S element `BEISPIEL2.C` in the library `EDT.BEISPIELE` then the above program can be compiled and linked with

```
/CALL-PROC CC.DO,(2).
```

The generated program can then be executed using

```
/START-EXECUTABLE-PROGRAM (E=BSP2C,L=EDT.BEISPIELE)
```

When `CC.DO` runs, the following or similar output is generated by the system or the compiler:

```
%  BLS0524 LLM 'SDFCC', VERSION '03.1A40' OF '2005-02-03 16:16:36' LOADED
%  BLS0551 COPYRIGHT (C) Fujitsu Siemens Computers GmbH 2005. ALL RIGHTS
RESERVED
%  CDR9992 : BEGIN C/C++(BS2000/OSD) VERSION 03.1A40
%  CDR9907 : NOTES: 0  WARNINGS: 0  ERRORS: 0  FATALS: 0
%  CDR9937 : MODULES GENERATED, CPU TIME USED = 2.6000 SEC
%  BND3102 SOME WEAK EXTERNS UNRESOLVED
%  BND1501 LLM FORMAT: '1'
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'UNRESOLVED EXTERNAL'
%  CDR9936 : END; SUMMARY: NOTES: 0  WARNINGS: 0  ERRORS: 0  FATALS: 0
%  CCM0998 CPU TIME USED: 3.6403 SECONDS
```

Let us assume that the file to be processed by this example program `EDT.RECHNUNG2` resembles the following:

```
00347563  00028654     1378.89  21.05.2007
00345781  00027349    21500.00  07.04.2007
00375863  00028937      248.23  19.05.2007
00242365  00012358     4577.54  23.03.2007
00416467  00046687     6776.31  10.05.2007
00576373  00015463      578.00  19.04.2007
00785214  00053417    65465.00  13.12.2006
00265432  00065743     6534.67  16.05.2007
```

```
00546315  00035476      656.34  29.05.2007
00675436  00015334     7878.45  04.05.2007
00353466  00087227      654.24  11.11.2006
00534267  00067854    52346.00  15.05.2007
00243535  00078921     4532.54  22.04.2007
00783432  00063223      548.19  17.05.2007
00623556  00054342     5346.32  17.03.2007
00234354  00065233     4534.65  08.05.2007
```

When `/START-EXECUTABLE-PROGRAM (E=BSP2C,L=EDT.BEISPIELE)` is called (on 4.5.2007), the following or similar output is displayed:

```
%  BLS0523 ELEMENT 'BSP2C', VERSION '@', TYPE 'L' FROM LIBRARY ':A:$USER.EDT
.BEISPIELE' IN PROCESS
%  BLS0524 LLM '$LIB-ELEM$EDT$BEISPIELE$$BSP2C$$', VERSION ' ' OF '2007-05-04
12
:22:24' LOADED
Start Beispiel2
Kdnr.: 00345781, Rechn.nr.: 00027349, Betrag:   21500.00 Euro
Kdnr.: 00242365, Rechn.nr.: 00012358, Betrag:    4577.54 Euro
Kdnr.: 00785214, Rechn.nr.: 00053417, Betrag:   65465.00 Euro
Kdnr.: 00353466, Rechn.nr.: 00087227, Betrag:     654.24 Euro
Kdnr.: 00623556, Rechn.nr.: 00054342, Betrag:    5346.32 Euro
Ende Beispiel  2
%  CCM0998 CPU TIME USED: 0.3276 SECONDS
```

7.3 Example 3 - C user routine

The following user routine is used to add numerical values in a column range in the current work file and insert the result in the last line.
The EDT functions `IEDTEXE`, `IEDTGET`, `IEDTPUT` and `IEDTPTM` are used.

```
/*****************************************************************/
/*                                                               */
/* Example 3                                                     */
/*                                                               */
/* This is an example of a user routine. It is declared          */
/* with USE COM='*',ENTRY=*,MODLIB=EDT.BEISPIELE .               */
/* Next, *sum <col1>-<col2> is used to call the SUM function     */
/* and the argument <col1>-<col2> is passed.                     */
/* The SUM function reads all the lines in the current work file */
/* and extracts the numerical values present in the <col1>-<col2>*/
/* column range. Lines containing invalid values are flagged     */
/* with mark 14 (i.e. they can be overwritten).                  */
/* The total value is entered in the last read line.             */
/*                                                               */
/* The example program performs the following individual         */
/* actions:                                                      */
/*                                                               */
/* 1) Determine the required column range from the call          */
/*    arguments.                                                 */
/* 2) Execute a loop in which all the lines in the current       */
/*    work area are read and the number present in the required  */
/*    column range is converted into a floating point value. If  */
/*    this is successful, add the floating point value to the total.*/
/*    If this is not successful, flag the line with mark 14      */
/*    (it can then be overwritten).                              */
/* 4) Once this loop has terminated, the total value is entered in */
/*    the last line and control returns to the caller.           */
/*                                                               */
/*****************************************************************/
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
/* Include files of the EDT subroutine interface */
#define EDT_V17
#include <iedtgle.h>
```

```
/* Alternative to the fill_buff function: Macro for filling the buffer */
                                    /* --------------------------- (1) */
#define IEDBUFF_FILL(buf,s) \
    (buf)->length = (strlen(s) + 4); \
    strncpy((char *)(buf)->text,s,(size_t)((buf)->length - 4))
/*********************************************************************/
/* Function: edtget                                                  */
/*                                                                   */
/* Task:                                                             */
/* This function uses the subroutine interface's IEDTGET function    */
/* to read a line from the current work file. The read operation     */
/* is always performed relative to the record with line number 0.    */
/*                                                                   */
/* Parameter: glcb:    (IN)  Pointer to the glcb supplied with       */
/*                           EDT.                                    */
/*            rec:     (IN)  Pointer to a data area in which         */
/*                           the line read by the GET function       */
/*                           is stored.                              */
/*            disp:    (IN)  Specifies the displacement of the       */
/*                           line to be read from line number 0.     */
/*            key      (OUT) Pointer to a field in which the read    */
/*                           key is stored                           */
/*********************************************************************/
static void
edtget(iedglcb *glcb,char *rec,int disp,iedbyte *key)
{
    iedamcb amcb = IEDAMCB_INIT;
    iedbyte key1[8] = {0,0,0,0,0,0,0,0};
    /* Enter values in IEDAMCB control block */
    IEDAMCB_SET_NO_MARKS(amcb);
    amcb.length_key_outbuffer = 8;
    amcb.length_rec_outbuffer = 256;
    amcb.length_key1 = 8;
    amcb.displacement = disp;  /* read <disp> records after 0 */
    /* Get current work file (from glcb) */
    strncpy((char *)amcb.filename,(char *)glcb->filename,8);
    IEDTGET(glcb,&amcb,key1,key,rec);
    rec[amcb.length_rec] = 0;  /* set C-string end */
}
/*********************************************************************/
/* Function: edtput                                                  */
/*                                                                   */
/* Task:                                                             */
/* This function uses the IEDTPUT function to write a record to      */
/* the location designated by key.                                   */
```

```
/*                                                                   */
/* Parameter: glcb:    (IN)  Pointer to the glcb supplied with       */
/*                           EDT.                                    */
/*            rec:     (IN)  Pointer to a data area in which        */
/*                           the record to be written is passed.     */
/*            key      (IN)  Pointer to a field in which the key     */
/*                           to be (over)written is located.         */
/* Return value: none                                                */
/*********************************************************************/
static void
edtput(iedglcb *glcb,char *rec,iedbyte *key)
{
    iedamcb amcb = IEDAMCB_INIT;
    amcb.length_key = 8;
    amcb.marks.mark_field = 0;
    amcb.length_rec = strlen(rec);
    /* Get current work file (from glcb) */
    strncpy((char *)amcb.filename,(char *)glcb->filename,8);
    IEDTPUT(glcb,&amcb,key,(unsigned char *)rec);
}
/*********************************************************************/
/* Function: edtptm                                                  */
/*                                                                   */
/* Task:                                                             */
/* This function uses the IEDTPTM function to write a mark           */
/* at the location designated by key.                                */
/*                                                                   */
/* Parameter: glcb:    (IN)  Pointer to the glcb supplied with       */
/*                           EDT.                                    */
/*            mark:    (IN)  Value of the mark that is to be written*/
/*            key      (IN)  Pointer to a field in which the key    */
/*                           for marking is located.                 */
/* Return value: none                                                */
/*********************************************************************/
static void
edtptm(iedglcb *glcb,int mark,iedbyte *key)
{
    iedamcb amcb = IEDAMCB_INIT;
    amcb.marks.mark_field = 0;
    amcb.length_key = 8;
    if (mark != 0)                /* ------------------------------ (2) */
        amcb.marks.mark_field = (1 << mark);
    /* Get current work file (from glcb) */
    strncpy((char *)amcb.filename,(char *)glcb->filename,8);
```

```
    IEDTPTM(glcb,&amcb,key);
}
/*********************************************************************/
/* Function: SUM                                                     */
/*                                                                   */
/* Task:                                                             */
/* This function uses the etdget function to read the entire         */
/* work file, extracts – for every line – the column range           */
/* that was passed as an argument and attempts to convert the        */
/* text present there into a floating point value.                   */
/* The identified numbers are added and the total value is           */
/* output in the last line (where it overwrites the specified        */
/* column range).                                                    */
/* Lines which do not contain a valid floating point value at the    */
/* specified location are made available for overwriting with        */
/* mark 14.                                                          */
/*                                                                   */
/* Parameter: glcb:    (IN)  Pointer to a glcb which is used for     */
/*                           IEDTGET or IEDTPUT calls                */
/*            cmd:     (IN)  Arguments with which sum was           */
/*                           called.                                 */
/* Return value: none                                                */
/*********************************************************************/
void
SUM(iedglcb *glcb,iedbuff *cmd)         /* --------------------------- (3) */
{
    char command[80];
    char line[256];
    char number[64];
    int len = cmd->length;
    size_t col1 = 0;
    size_t col2 = 0;
    iedbyte key[8];
    float fnum;
    float sum;
    int disp;
    /* Determine column range from call argument */
    cmd->text[len] = '\0'; /* Set end character for C string */
    if (sscanf((char *)cmd->text," %d - %d ",&col1,&col2) < 2)
    {
        glcb->IEDGLCB_RC_MAINCODE = IEDGLCBcmd_unrec_user_error;
        glcb->IEDGLCB_RC_SUBCODE1 = IEDGLCBparameter_error;
        return;                      /*----------------------------- (4) */
    }
    /* @PAR PROT=ON statement, therefore mark 14 effective */
    IEDBUFF_FILL((iedbuff *)command,"@PAR PROT=ON");
```

```
    IEDTEXE(glcb,(iedbuff *)command);
    sum = 0.0;
    /* Read all lines in current work file */
    for (disp = 1; disp < 99999999; disp++)  /*----------------------- (5) */
    {
        /* Read next line */
        edtget(glcb,(char *)&line,disp,key);
        if (glcb->IEDGLCB_RC_MAINCODE != 0)
            return; /* Pass on MAINCODE unchanged */
        /* If reading goes beyond the last line number then */
        /* IEDTGET returns "last record"                    */
        if (glcb->IEDGLCB_RC_SUBCODE1 == IEDGLCBlast_record)
            break; /* Exit loop */
        /* Extract numerical value */
        strncpy(number,&line[col1 - 1],col2 - col1 + 1);
        number[col2 - col1 + 1] = '\0';
        if (sscanf(number," %f ",&fnum) >= 1)
        {
            sum += fnum;
            edtptm(glcb,0,key); /* Reset mark if necessary */
        }
        else /* No valid floating point number in range */
        {
            /* Set mark 14 – line can be overwritten */
            edtptm(glcb,14,key);
        }
    }
    /* Set total value in last line and write it back */
    sprintf(number,"%6.2f      ",sum);
    strncpy(&line[col1 - 1],number,col2 - col1 + 1);
    edtput(glcb,(char *)line,key);
}
/*********************************************************************/
/* Function: SUM@I                                                   */
/*                                                                   */
/* Task:                                                             */
/* This is the initialization routine for SUM.                       */
/*                                                                   */
/* Parameter: glcb:    (IN)  Pointer to a glcb which is used, for    */
/*                           example, to specify the character set   */
/*                           to be used when sum is called.          */
/*                                                                   */
/* Return value: none                                                */
/*********************************************************************/
```

```
void SUM@I(iedglcb *glcb)               /* -------------------------- (6) */
{
    glcb->indicator.compatible_format = 1;
    memcpy(glcb->ccsn,"EDF04F    ",8); /* sum expects EDF04F */
    glcb->rc.rc_nbr = 0;
}
```

Explanations

(1) As an alternative to the `fill_buff` function from examples 1 and 2, a macro declared with `#define` is used here to fill the statement buffer.

(2) The `edtptm` function sets only one mark in each case. To delete a mark, it must be called with the value `0`.

(3) The function name is defined in uppercase because the user statement is to be declared with @USE COM='*',ENTRY=*,... and called with `*sum c1 - c2`. In this case, EDT converts the first part of the user statement (the statement name) into uppercase characters (see section "Calling a user-defined statement").

(4) If the call argument cannot be converted into two integer values (start and end column) then the user routine is exited with a return code. The message `EDT5410` is then displayed. Naturally, other checks should be incorporated here
(e.g. `col1 < col2`) and the corresponding message texts should be written.
For enhanced clarity, this has not been done in the example.

(5) The method for the sequential reading of the work file implemented here is not particularly effective since it is always necessary to restart the read operation from the beginning again. Readers should consider how it might be possible to develop a more effective algorithm by using the key returned in `key` with a constant `amcb.displacement = +1.`

(6) An initialization routine has been implemented so that the call of the user routine takes place via the V17 interface using the `EDF04F` character set.

If the procedure explained in the section "Producing user routines in C" is stored in a file named `CCMOD.DO` in BS2000 and the source file is stored as the S element `ANWEND.C` in the library `EDT.BEISPIELE` then the above program can be compiled with

```
/CALL-PROC CCMOD.DO,(1)
```

and `LLM ANWEND1` can be stored in the library `EDT.EXAMPLES`.

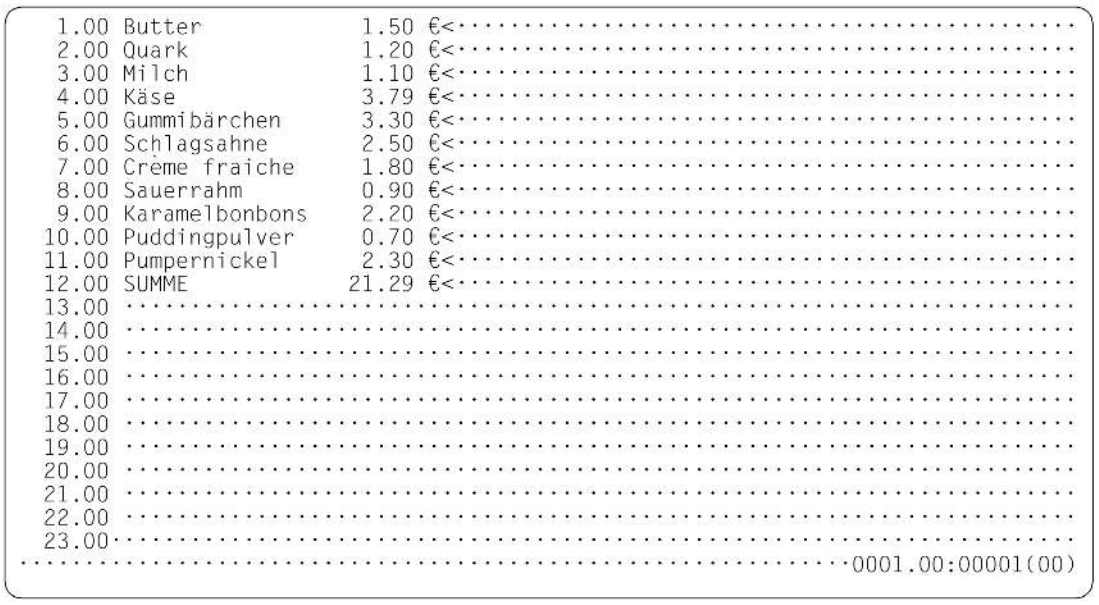
If the user statement is declared with

@USE COMMAND='*',ENTRY=*,MODLIB=EDT.BEISPIELE

then the `*SUM` statement can be used, for example, to add prices in a purchasing list:

```
  1.00 Butter           1.50 €<·····
  2.00 Quark            1.20 €<·····
  3.00 Milch            1.10 €<·····
  4.00 Käse             3.79 €<·····
  5.00 Gummibärchen     3.30 €<·····
  6.00 Schlagsahne      2.50 €<·····
  7.00 Crème fraiche    1.80 €<·····
  8.00 Sauerrahm        0.90 €<·····
  9.00 Karamelbonbons   2.20 €<·····
 10.00 Puddingpulver    0.70 €<·····
 11.00 Pumpernickel     2.30 €<·····
 12.00 SUMME           00.00 €<·····
 13.00 ·····
 14.00 ·····
 15.00 ·····
 16.00 ·····
 17.00 ·····
 18.00 ·····
 19.00 ·····
 20.00 ·····
 21.00 ·····
 22.00 ·····
 23.00·····
*sum 16-21·····0001.00:00001(00)
```

The following screen is displayed when `*sum 16-21` is entered:


1.00 Butter 1.50 €<
2.00 Quark 1.20 €<
3.00 Milch 1.10 €<
4.00 Käse 3.79 €<
5.00 Gummibärchen 3.30 €<
6.00 Schlagsahne 2.50 €<
7.00 Crème fraiche 1.80 €<
8.00 Sauerrahm 0.90 €<
9.00 Karamelbonbons 2.20 €<
10.00 Puddingpulver 0.70 €<
11.00 Pumpernickel 2.30 €<
12.00 SUMME 21.29 €<
13.00
14.00
15.00
16.00
17.00
18.00
19.00
20.00
21.00
22.00
23.00
0001.00:00001(00)

7.4 Example 4 - C main program and user routine in a single source

This example illustrates the combination of a C main program and a C user routine in the same source.
Because when the @USE statement is run, EDT first searches for the specified entry with VSVI, it finds the entry in the main program and uses this.
No dynamic loading is therefore performed.
In the example, the @USE statement is issued directly in the main program, with the result that the user statement *rot (or similar.) is immediately available to the user.
Since the user routine does not expect any special statement string, example 4 can also act as an example of a user routine that can be called with @RUN ENTRY=ROT13.

```
/*****************************************************************/
/*                                                               */
/* Example 4                                                     */
/*                                                               */
/* This example illustrates the combination of a C main program  */
/* with a user routine. It makes it possible to pass a           */
/* file name as an argument. This file is read in before switching */
/* to the screen dialog. The user routine in the                 */
/* same program encrypts / decrypts all the records in the current */
/* work file using the (primitive) ROT13 procedure.              */
/*                                                               */
/* The main program performs the following actions:              */
/*                                                               */
/* 1) Determine the file name from the call argument.            */
/* 2) Read the file using an @OPEN statement via IEDTCMD         */
/* 3) Set up the user statement "*ROT13" with @USE               */
/* 4) Branch to the screen dialog with @DIALOG                   */
/*                                                               */
/* The user routine ROT13 performs the following actions:        */
/*                                                               */
/* 1) Read all the records in the current work file in a loop.   */
/* 2) Encrypt each record with ROT13.                            */
/* 3) Write the record back to the work file with IEDTPUT        */
/*                                                               */
/*****************************************************************/
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
```

```
/* Include files of the EDT subroutine interface */
#define EDT_V17
#include <iedtgle.h>
/* Create and initialize the EDT subroutine interface data */
/* structures required for this example.                   */
static iedglcb glcb = IEDGLCB_INIT;
static iedupcb upcb = IEDUPCB_INIT;
static iedbuff *command = NULL;
static iedbuff *message1 = NULL;
static iedbuff *message2 = NULL;
/*****************************************************************/
/* Function: printrc                                             */
/*                                                               */
/* Task:                                                         */
/* If EDT has returned an error message, then the error message  */
/* is output by this function.                                   */
/*                                                               */
/* Parameter: errmsg   (IN)  Pointer to the additional error     */
/*                           message to be output if an error occurs*/
/*                                                               */
/* Return value: none                                            */
/*****************************************************************/
static void
printrc(char *errmsg)
{
    char message[81];
    if ((glcb.rc.structured_rc.mc.maincode != 0) &&
         (glcb.return_message.structured_msg.rmsgl > 0))
    {
        printf("%s\n",errmsg);   /* Output the passed error message */
        strncpy(message,(char*)glcb.return_message.structured_msg.rmsgf,
            glcb.return_message.structured_msg.rmsgl);
        message[glcb.return_message.structured_msg.rmsgl] = 0x00;
        printf("Meldungstext: %s\n",message);  /* Output EDT message */
        exit(1);
    }
}
```

```
/*****************************************************************/
/* Function: fill_buff                                           */
/*                                                               */
/* Task:                                                         */
/* This function enters content and the record length field in a */
/* record of variable length.                                    */
/*                                                               */
/* Parameter: p:       (IN)  Pointer to a structure of type      */
/*                           iedbuff, which contains the variable*/
/*                           length record to be set.            */
/*            textp:   (IN)  Points to a string which contains the*/
/*                           text to be entered. The length      */
/*                           of the string implicitly defines the*/
/*                           record length (length of string + 4).*/
/*                                                               */
/* Return value: none                                            */
/*****************************************************************/
static void
fill_buff(iedbuff *p,char *textp)
{
    size_t l_text;                          /* Length of string    */
    if ((l_text = strlen(textp)) > 2044)
        l_text = 2044;     /* Restrict length to 2044 characters   */
    strncpy((char *)p->text,textp,l_text);  /* Enter text          */
    p->length = l_text + 4;                 /* Enter record length */
}
/*****************************************************************/
/* Function: edtcmd                                              */
/*                                                               */
/* Task:                                                         */
/* This function enters the passed strings in records of         */
/* variable lengths (DMS format) and then calls the EDT's        */
/* CMD interface.                                                */
/*                                                               */
/* Parameter: cmd:     (IN)  Pointer to a string which contains  */
/*                           the EDT statement(s) that are to be */
/*                           executed. The length of the string  */
/*                           implicitly defines the record length*/
/*                           (String length + 4).                */
/*            msg1:    (IN)  Pointer to a string which contains the */
/*                           text to be entered. The length      */
/*                           of the string implicitly defines the*/
/*                           record length (length of string + 4).*/
/*            msg2:    (IN)  Pointer to a string which contains the */
/*                           text to be entered. The length      */
/*                           of the string implicitly defines the*/
```

```
/*                          record length (length of string + 4).  */
/*                                                                 */
/* Return value: none                                              */
/*******************************************************************/
static void
edtcmd(char *cmd,char *msg1,char *msg2)
{
    fill_buff(command,cmd);
    fill_buff(message1,msg1);
    fill_buff(message2,msg2);
    IEDTCMD(&glcb,&upcb,command,message1,message2);
}
/*******************************************************************/
/* Function: edtget                                                */
/*                                                                 */
/* Task:                                                           */
/* This function uses the subroutine interface's GET function      */
/* to read a line from the current work file. The read operation   */
/* is performed relative to the record with line number 0.         */
/*                                                                 */
/* Parameter: glcb:    (IN)  Pointer to the glcb supplied with     */
/*                           EDT.                                  */
/*            rec:     (IN)  Pointer to a data area in which       */
/*                           the line read by the GET function     */
/*                           is stored.                            */
/*            disp:    (IN)  Specifies the displacement of the     */
/*                           line to be read from line number 0.   */
/*            key      (OUT) Pointer to a field in which the read  */
/*                           key is stored                         */
/*******************************************************************/
static void
edtget(iedglcb *glcb,char *rec,int disp,iedbyte *key)
{
    iedamcb amcb = IEDAMCB_INIT;
    iedbyte key1[8] = {0,0,0,0,0,0,0,0};
    /* Enter values in IEDAMCB control block */
    IEDAMCB_SET_NO_MARKS(amcb);
    amcb.length_key_outbuffer = 8;
    amcb.length_rec_outbuffer = 2044;
    amcb.length_key1 = 8;
    amcb.displacement = disp;  /* read <disp> records after 0 */
    /* Get current work file (from glcb) */
    strncpy((char *)amcb.filename,(char *)glcb->filename,8);
    IEDTGET(glcb,&amcb,key1,key,rec);
    rec[amcb.length_rec] = 0;  /* set C-string end */
}
```

```
/*********************************************************************/
/* Function: edtput                                                  */
/*                                                                   */
/* Task:                                                             */
/* This function uses the IEDTPUT function to write a record to     */
/* the location designated by key.                                   */
/*                                                                   */
/* Parameter: glcb:    (IN)  Pointer to the glcb supplied with      */
/*                           EDT.                                    */
/*            rec:     (IN)  Pointer to a data area in which        */
/*                           the record to be written is passed.     */
/*            key      (IN)  Pointer to a field in which the key     */
/*                           to be (over)written is located.         */
/* Return value: none                                                */
/*********************************************************************/
static void
edtput(iedglcb *glcb,char *rec,iedbyte *key)
{
    iedamcb amcb = IEDAMCB_INIT;
    amcb.length_key = 8;
    amcb.marks.mark_field = 0;
    amcb.length_rec = strlen(rec);
    strncpy((char *)amcb.filename,(char *)glcb->filename,8);
    IEDTPUT(glcb,&amcb,key,(unsigned char *)rec);
}
/* Static data for encryption */
static char* lcc = "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz";
static char* ucc = "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ";
/*********************************************************************/
/* Function: ROT13                                                   */
/*                                                                   */
/* Task:                                                             */
/* This function uses the EDTGET function to read the entire         */
/* work file and encrypts every line using the ROT13 encryption      */
/* procedure (i.e. all the letters are moved 13 characters)          */
/* As a result, performing encryption twice restores the original    */
/*                                                                   */
/* Parameter: glcb:    (IN)  Pointer to a glcb which is used for     */
/*                           IEDTGET or IEDTPUT                      */
/*            cmd:     (IN)  Arguments with which ROT13 was          */
/*                           called (not used).                      */
/* Return value: none                                                */
/*********************************************************************/
```

```
void
ROT13(iedglcb *glcb,iedbuff *cmd)
{
    char line[2048];
    iedbyte key[8];
    int  disp;
    /* Read all lines in current work file */
    for (disp = 1; disp < 99999999; disp++)   /*-------------------- (1) */
    {
        unsigned int j = 0;
        /* Read next line */
        edtget(glcb,(char *)&line,disp,key);
        if (glcb->IEDGLCB_RC_MAINCODE != 0)  /* -------------------- (2) */
        {
            glcb->IEDGLCB_RC_MAINCODE = IEDGLCBcmd_runtime_error;
           return; /* Any message from IEDTGET remains in place */
        }
        /* If reading goes beyond the last line number then */
        /* IEDTGET returns "last record"                    */
        if (glcb->IEDGLCB_RC_SUBCODE1 == IEDGLCBlast_record)
            break; /* Exit loop */
        /* Encrypt the line */
        for (j = 0; j < strlen((char *)&line); j++)
        {
            char* pos;
            char ch = line[j];
            if (isalpha(ch))
            {
                if (islower(ch))
                    pos = index(lcc,ch);
                else
                    pos = index(ucc,ch);
                pos += 13;
                line[j] = *pos;
            }
        }
        /* Write the line back */
        edtput(glcb,(char *)&line,key);
        if (glcb->IEDGLCB_RC_MAINCODE != 0)
        {
            glcb->IEDGLCB_RC_MAINCODE = IEDGLCBcmd_runtime_error;
            return; /* Any message from IEDTPUT remains in place */
        }
    }
```

```
    /* Reset subcode so that no message is issued */
    glcb->IEDGLCB_RC_SUBCODE1 = 0;     /* ------------------------- (3) */
}
/*****************************************************************/
/* Function: ROT13@I                                             */
/*                                                               */
/* Task:                                                         */
/* This is the initialization routine for ROT13.                 */
/*                                                               */
/* Parameter: glcb:    (IN)  Pointer to a glcb which is used, for*/
/*                           example, to specify the character set*/
/*                           to be used when ROT13 is called.    */
/*                                                               */
/* Return value: none                                            */
/*****************************************************************/
void
ROT13@I(iedglcb *glcb)
{
    glcb->indicator.compatible_format = 1;
    memcpy(glcb->ccsn,"EDF04F    ",8); /* ROT13 expects EDF04F */
    glcb->rc.rc_nbr = 0;
}
/*****************************************************************/
/* Main program                                                  */
/*****************************************************************/
int
main(int argc,char *argv[])
{
    char cmd[257];      /* Area for the construction of EDT statements */
    int opt;
    extern int optind, opterr, optopt;
    extern char *optarg;
    /* Evaluate call parameters */
    while ((opt = getopt(argc,argv,"f:x:")) != -1)    *------------- (4) */
    {
        switch (opt)
        {
        case 'f':
            sprintf(cmd,"@OPEN FILE=%s",optarg);
            break;
        case 'x':
            sprintf(cmd,"@OPEN POSIX-FILE=%s",optarg);
            break;
        case ':':
            printf("Argument für -%c fehlt",optopt);
```

```
            return 0;
        default:
            printf("Aufruf mit -f <file> oder -x <posix-file>");
            return 0;
        }
    }
    /* Provide buffer */
    command = (iedbuff *)malloc(2048);
    message1 = (iedbuff *)malloc(2048);
    message2 = (iedbuff *)malloc(2048);
    edtcmd(&(cmd[0]),"","");
    printrc("Fehler beim Einlesen der Datei.");
    /* Set up user statement. MODLIB does not have to be specified.*/
    edtcmd("USE COM='*',ENTRY=ROT13","","");    /* ----------------- (5) */
    printrc("Fehler bei der @USE-Anweisung.");
    edtcmd("SETF(0);DIALOG","Beispiel 4 fuer die UP-Schnittstelle","");
    printrc("Fehler bei der @SETF- oder der @DIALOG-Anweisung!");
    edtcmd("HALT","","");
    return 0;                                    /* -------------- (6) */
}
```

Explanations:

(1) The method for the sequential reading of the work file implemented here is not particularly effective since it is always necessary to restart the read operation from the beginning again. Readers should consider how it might be possible to develop a more effective algorithm by using the key returned in `key` with a constant `amcb.displacement = +1`.

(2) For reasons of clarity, there is only a rudimentary implementation of the error handling mechanisms. At this point, a check should at least be performed to determine whether EDT has returned a message and, if necessary, issue a message yourself (enter with the original EDT return code).

(3) An unexpected return code always results in the output of the message `EDT5410` following return from the user routine.

(4) The call parameters `-f <filename>` and `-x <posix-filename>` are accepted.

(5) The user statement is set up before the switchover to the screen dialog. Since the entry is located in the program itself, it is not necessary to specify a library. Since the `ROT13` function does not evaluate the passed statement string, it can be called in any desired way, for example with `*rot` or even just with `*`. A call with @RUN E=ROT13 is therefore also possible.

(6) If the program is to be used “productively”, a check for any files that are still open and a corresponding close dialog must be added here.

If the procedure explained in the section “Producing main programs in C” is stored in a file named `CC.DO` in BS2000 and the source file is stored as the S element `BEISPIEL4.C` in the library `EDT.BEISPIELE` then the above program can be compiled and linked with

```
/CALL-PROC CC.DO,(4).
```

The generated program can then be executed, for example, using

```
/ST-EX-P (edt.beispiele,bsp4c),p-p='-f edt.test4'
```

This representation is deliberately highly abbreviated in order to demonstrate that this type of starter program can also be used in BS2000 to start EDT relatively easily in order to edit a given file.

When CC.DO runs, the following or similar output is generated by the system or the compiler:

```
%  BLS0524 LLM 'SDFCC', VERSION '03.1A40' OF '2005-02-03 16:16:36' LOADED
%  BLS0551 COPYRIGHT (C) Fujitsu Siemens Computers GmbH 2005. ALL RIGHTS
RESERVED
%  CDR9992 : BEGIN C/C++(BS2000/OSD) VERSION 03.1A40
%  CDR9907 : NOTES: 1  WARNINGS: 0  ERRORS: 0  FATALS: 0 -------------- (1)
%  CDR9937 : MODULES GENERATED, CPU TIME USED = 3.3800 SEC
%  BND3102 SOME WEAK EXTERNS UNRESOLVED
%  BND1501 LLM FORMAT: '1'
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'UNRESOLVED EXTERNAL'
%  CDR9936 : END; SUMMARY: NOTES: 1  WARNINGS: 0  ERRORS: 0  FATALS: 0
%  CCM0998 CPU TIME USED: 4.6826 SECONDS
```

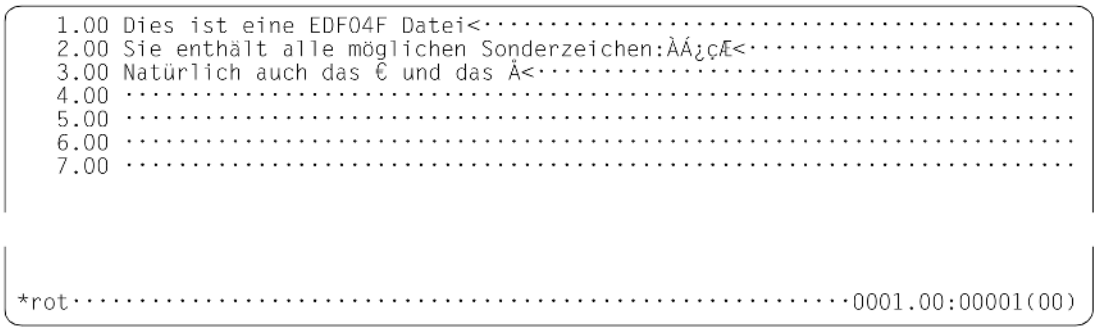
Explanations

(1) The "NOTE" occurs because the call parameter cmd is not addressed by ROT13.

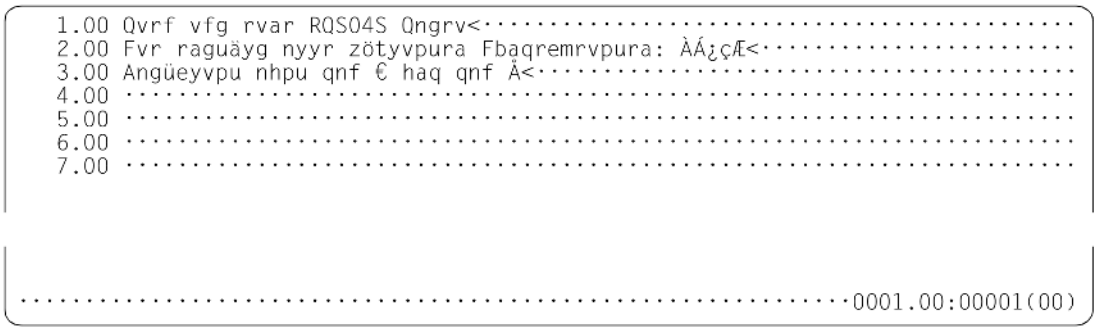
Let us assume that the file to be processed by this example program EDT.TEST4 has the CCS EDF04F and resembles the following:

```
Dies ist eine EDF04F Datei.
Sie enthält alle möglichen Sonderzeichen: ÀÁ¿çÆ.
Natürlich auch das € und das Å.
```

When the program is called, the following screen is displayed:



After entry of the *rot user statement, the output is as follows:



7.5 Example 5 - Assembler main program

In terms of functionality, example 5 is identical to example 1, the difference being that it is programmed in Assembler instead of C.

```
          TITLE 'BEISPIEL5'
*********************************************************************
*                                                                   *
* Example 5                                                         *
*                                                                   *
* This example uses only the iedtcmd                                *
* interface to execute EDT statements.                              *
* The functionality is identical to that of "Example 1" in C.       *
*                                                                   *
* The example program performs the following actions:               *
*                                                                   *
* 1) Read a selection criterion (CCSN)                              *
* 2) Output a table of contents to work file 0                      *
*    using the @SHOW statement (Format 1).                          *
* 3) Delete all the lines which do not contain the search           *
*    criterion with the @ON statement (Format 10).                  *
* 4) Set work file 0 and then switch to the F mode                  *
*    dialog by means of a @SETF statement followed by a             *
*    @DIALOG statement.                                             *
* 5) The user can now edit the lines that are output                *
*    and after that terminate EDT and consequently also this        *
*    example program.                                               *
*                                                                   *
*********************************************************************
*
BEISP5   START
BEISP5   AMODE ANY
BEISP5   RMODE ANY
         GPARMOD 31
*
         BALR  R10,0
         USING *,R10
*
*        OUTPUT START MESSAGE
*
         LA    R1,6               * LENGTH = 6 FOR THE OUTPUT
         STH   R1,LEERMSG         * OF A BLANK
*
         WROUT LEERMSG,WROUTERR
         WROUT STARTMSG,WROUTERR
         WROUT LEERMSG,WROUTERR
```

```
*
*        READ IN CCSN
*
         WRTRD PROMPT,,EINB,,12,WRTRDERR
*
         LH    R15,EINB           LENGTH OF INPUT
         SH    R15,=Y(5)          4 BYTES FOR SLF AND 1 FOR EX
         EX    R15,EXMVCCSN       PASS CCS TO THE ON STATEMENT
         LA    R14,CCSN           SET POSITION AT CHARACTER
         LA    R14,1(R14,R15)     AFTER CCSN
         MVI   0(R14),''''        SET APOSTROPHE          ---------------- (1)
*
         LA    R1,4               * LENGTH = 4 TO PASS AN
         STH   R1,LEERMSG         * EMPTY STRING TO EDT
*
*        OUTPUT TABLE OF CONTENTS TO WORK FILE 0
*
         LA    R1,CMD1            ADDRESS OF EDT STATEMENT
         LA    R2,LEERMSG         MSG1 = EMPTY STRING
         LA    R3,LEERMSG         MSG2 = EMPTY STRING
         LA    R4,ZUSMSG1         ADDR. OF ERROR MESSAGE TO BE OUTPUT
*
         BAL   R11,CMDCALL        CALL OF THE EDT'S CMD INTERFACE
*
*       DELETE ALL LINES WHICH DO NOT CORRESPOND TO THE SEARCH
*       CRITERION (CCSN) AND RENUMBER THE REMAINING LINES
*
         LA    R1,CMD2            ADDRESS OF THE EDT STATEMENT
         LA    R2,LEERMSG         MSG1 = EMPTY STRING
         LA    R3,LEERMSG         MSG2 = EMPTY STRING
         LA    R4,ZUSMSG2         ADDR. OF ERROR MESSAGE TO BE OUTPUT
*
         BAL   R11,CMDCALL        CALL OF THE EDT'S CMD INTERFACE
*
*        GO TO WORK FILE 0 AND SWITCH TO THE
*        F MODE DIALOG
*
         LA    R1,CMD3            ADDRESS OF EDT STATEMENT
         LA    R2,MESSAGE3        ADDRESS OF MESSAGE TO BE OUTPUT
         LA    R3,LEERMSG         MSG2 = EMPTY STRING
         LA    R4,ZUSMSG3         ADDR. OF ERROR MESSAGE TO BE OUTPUT
*
         BAL   R11,CMDCALL        CALL OF THE EDT'S CMD INTERFACE
*
```

```
*        OUTPUT END MESSAGE
*
         LA    R1,6               * LENGTH = 6 FOR THE OUTPUT
         STH   R1,LEERMSG         * OF A BLANK
*
         WROUT LEERMSG,WROUTERR
         WROUT ENDEMSG,WROUTERR
         WROUT LEERMSG,WROUTERR
*
         TERM                           TERMINATE PROGRAM
*
*        HANDLING OF EDT ERRORS
*
EDTERR   EQU   *
         OC    EGLRMSGL,EGLRMSGL        LENGTH OF EDT MESSAGE = 0 ?
         BZ    NOMSG                    YES, THEN OUTPUT NOTHING
*
*        OUTPUT ADDITIONALLY SPECIFIED MESSAGE
*
         LH    R5,0(R4)                 LENGTH OF ADDITIONAL MESSAGE
         BCTR  R5,0                     -1 FOR EX INSTRUCTION
         EX    R5,EXMVC                 TAKE OVER MESSAGE
*
         WROUT ZUSMSG,WROUTERR          OUTPUT ADDITIONAL MESSAGE
*
*        OUTPUT EDT ERROR MESSAGE
*
         LH    R1,EGLRMSGL              * LENGTH OF EDT MESSAGE
         LA    R1,5+14(R1)              * + 5 (DUE TO RECORD LENGTH FIELD)
*                                       * + 14 (DUE TO "MELDUNGSTEXT: ")
         STH   R1,ERRMSG1               * STORE IN LENGTH FIELD
         MVC   ERRTEXT,EGLRMSGF         PASS MESSAGE TEXT
*
         WROUT ERRMSG1,WROUTERR         OUTPUT EDT MESSAGE
*
NOMSG    EQU   *
         TERM                           TERMINATE PROGRAM
*
*        HANDLING OF WRTRD ERRORS
*
WRTRDERR EQU   *
         WROUT ERRMSG2,WROUTERR
*
WROUTERR EQU   *
         TERM                           TERMINATE PROGRAM
         EJECT
```

```
*********************************************************************
*                                                                   *
*        SUBROUTINES                                                *
*                                                                   *
*********************************************************************
*
*********************************************************************
* SUBROUTINE: CMDCALL
*
*                                                                   *
* TASK:                                                             *
* THIS SUBROUTINE ENTERS VALUES IN THE PARAMETER LIST FOR THE EDT'S  *
* INTERFACE AND THEN CALLS THE CMD INTERFACE                        *
* AFTER  A SUCCESSFUL RETURN FROM EDT, CONTROL IS RETURNED TO THE   *
* CALLER. IF AN ERROR OCCURS, CONTROL PASSES TO AN ERROR ROUTINE    *
* WHICH OUTPUTS THE ADDITIONALLY PASSED ERROR MESSAGE AS WELL       *
* AS THE MESSAGE RETURNED BY EDT.                                   *
*                                                                   *
* PARAMETER: (R1):  (IN) ADDRESS OF A RECORD OF VARIABLE LENGTH     *
*                        (DMS FORMAT) WHICH CONTAINS THE EDT        *
*                        STATEMENTS THAT ARE TO BE EXECUTED.        *
*            (R2):  (IN) ADDRESS OF A RECORD OF VARIABLE LENGTH     *
*                        (DMS FORMAT) WHICH CONTAINS THE MESSAGE1   *
*                        THAT IS TO BE OUTPUT BY EDT.               *
*            (R3):  (IN) ADDRESS OF A RECORD OF VARIABLE LENGTH     *
*                        (DMS FORMAT) WHICH CONTAINS THE MESSAGE2   *
*                        THAT IS TO BE OUTPUT BY EDT.               *
*            (R4):  (IN) ADDRESS OF A RECORD OF VARIABLE LENGTH     *
*                        (DMS FORMAT) WHICH CONTAINS THE ADDITIONAL *
*                        MESSAGE FOR OUTPUT BY EDT IF AN ERROR OCCURS*
*            (R11): (IN) RETURN ADDRESS                             *
*                                                                   *
* RETURN VALUE: NONE                                                *
*********************************************************************
*
CMDCALL  EQU   *
         STM   R1,R3,COMMAND           FILL CMD PARAMETER LIST
         LA    R1,CMDPL                ADDR. OF CMD PARAMETER LIST
         LA    R13,SAVEAREA            ADDR. OF SAVE AREA
         L     R15,=V(IEDTCMD)         ADDR. OF CMD INTERFACE
*
         BALR  R14,R15                 CALL EDT CMD INTERFACE
*
         CLC   EGLMRET,=AL2(EUPRETOK)  ERROR ON EDT CALL?
         BNE   EDTERR                  YES? OUTPUT ERROR MESSAGE
         BR    R11                     NO? RETURN TO CALLER
         EJECT
```

```
*********************************************************************
*                                                                   *
*        CONSTANTS                                                  *
*                                                                   *
*********************************************************************
*
*        REGISTER DEFINITIONS
*
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         EJECT
*********************************************************************
*                                                                   *
*        FIELDS                                                     *
*                                                                   *
*********************************************************************
*
SAVEAREA DS    18F
*
EXMVC    MVC   ZUSMSG(0),0(R4)         EX INSTRUCTION FOR TRANSFER
*                                      OF THE ADDITIONAL MESSAGE
*
EXMVCCSN MVC   CCSN(0),EINGABE         ENTER CCSN IN ON STATEMENT
LEERMSG  DC    Y(LEERMEND-LEERMSG)     EMPTY RECORD
         DS    CL2
         DC    ' '
         DC    ' '
LEERMEND EQU   *
*
PROMPT   DC    Y(PRMPTEND-PROMPT)      INPUT PROMPT
         DS    CL2
         DC    C' '
         DC    C'Bitte CCSN eingeben (UTFE, UTF16, EDF...): '
PRMPTEND EQU   *
```

```
*
EINB     DC    Y(EINBEND-EINB)         INPUT RANGE FOR WRTRD
         DS    CL2
EINGABE  DC    C'        '
EINBEND  EQU   *
*
CMD1     DC    Y(CMD1END-CMD1)         EDT STATEMENT SEQUENCE 1
         DS    CL2
         DC    C'SHOW F=* TO 1 LONG'
CMD1END  EQU   *
*
CMD2     DC    Y(CMD2END-CMD2)         EDT STATEMENT SEQUENCE 2
         DS    CL2
         DC    C'ON &&:100-107: FIND NOT '''
CCSN     DC    C'        '
         DC    C'  DELETE;RENUMBER'
CMD2END  EQU   *
*
CMD3     DC    Y(CMD3END-CMD3)         EDT STATEMENT SEQUENCE 3
         DS    CL2
         DC    C'SETF(0);DIALOG'
CMD3END  EQU   *
*
MESSAGE3 DC    Y(MSG3END-MESSAGE3)     MESSAGE TO BE OUTPUT BY EDT
         DS    CL2
         DC    C'Beispiel 5 fuer die UP-Schnittstelle'
MSG3END  EQU   *
*
ERRMSG1  DC    Y(ERRM1END-ERRMSG1)     RECORD FOR OUTPUT OF EDT MESSAGE
         DS    CL2
         DC    C' '
         DC    C'Meldungstext: '
ERRTEXT  DS    CL80
ERRM1END EQU   *
*
ERRMSG2  DC    Y(ERRM2END-ERRMSG2)     ERROR MESSAGE ON INPUT ERRORS
         DS    CL2
         DC    C' '
         DC    C'Eingabe zu lang!'
ERRM2END EQU   *
*
STARTMSG DC    Y(STRTMEND-STARTMSG)    START MESSAGE
         DS    CL2
         DC    C' '
         DC    C'Start Beispiel5'
STRTMEND EQU   *
*
ENDEMSG  DC    Y(ENDEMEND-ENDEMSG)     END MESSAGE
```

```
         DS    CL2
         DC    C' '
         DC    C'Ende Beispiel5'
ENDEMEND EQU   *
*
ZUSMSG1  DC    Y(ZUSM1END-ZUSMSG1)     ADDITIONAL ERROR MESSAGE
         DS    CL2                     FOR OUTPUT IN EDT
         DC    C' '                    STATEMENT SEQUENCE 1
         DC    C'Fehler bei der @SHOW-Anweisung!'
ZUSM1END EQU   *
*
ZUSMSG2  DC    Y(ZUSM2END-ZUSMSG2)     ADDITIONAL ERROR MESSAGE
         DS    CL2                     FOR OUTPUT IN EDT
         DC    C' '                    STATEMENT SEQUENCE 2
         DC    C'Fehler bei der @ON- oder der @RENUMBER-Anweisung!'
ZUSM2END EQU   *
*
ZUSMSG3  DC    Y(ZUSM3END-ZUSMSG3)     ADDITIONAL ERROR MESSAGE
         DS    CL2                     FOR OUTPUT IN EDT
         DC    C' '                    STATEMENT SEQUENCE 3
         DC    C'Fehler bei der @SETF- oder der @DIALOG-Anweisung!'
ZUSM3END EQU   *
*
ZUSMSG   DC    Y(ZUSMEND-ZUSMSG)       RECORD OF VARIABLE LENGTH
         DS    CL2                     FOR OUTPUT OF THE ADDITIONAL
         DC    C' '                    ERROR MESSAGE
         DC    CL80' '
ZUSMEND  EQU   *
*
*        PARAMETER LIST FOR THE EDT CMD INTERFACE
*
CMDPL    DC    A(EDTGLCB)              EDTGLCB ADDRESS
         DC    A(EDTUPCB)              EDTUPCB ADDRESS
COMMAND  DC    A(0)                    ADDRESS OF EDT STATEMENTS
MSG1     DC    A(0)                    ADDRESS OF MESSAGE1
MSG2     DC    A(0)                    ADDRESS OF MESSAGE2
*
*        EDT-SPECIFIC INTERFACE MACROS IN V17.0A
*
         IEDTGLCB C,VERSION=2          ------------------------------- (2)
*
         IEDTUPCB C,VERSION=3
         END
```

Explanations

(1) There must not be any blanks between `CCSN` and the apostrophe as otherwise no search for substrings (e.g. `EDF`) is possible.

(2) The V17 version for the relevant interface is generated.

If the procedure explained in the section “Producing main programs in Assembler” is stored in a file named `ASS.DO` in BS2000 and the source file is stored as the S element `BEISPIEL5.ASS` in the library `EDT.BEISPIELE` then the above program can be compiled and linked with

```
/CALL-PROC ASS.DO,(5)
```

The generated program can then be executed using

```
/START-EXECUTABLE-PROGRAM (E=BSP5A,L=EDT.BEISPIELE)
```

When ASS.DO runs, the following or similar output is generated by the system or the compiler:

```
%  BLS0523 ELEMENT 'ASSEMBH', VERSION '012', TYPE 'C' FROM LIBRARY' :MARS:
$TSOS.SYSPRG.ASSEMBH.012' IN PROCESS
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '01.2C00' OF '2002-03-06' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 2002. ALL RIGHTS
RESERVED
%  ASS6010 V01.2C00 OF BS2000 ASSEMBH  READY
%  ASS6011 ASSEMBLY TIME: 836 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 291 MSEC
%  ASS6012 END OF ASSEMBH
%  BND0500 BINDER VERSION 'V02.3A00' STARTED
%  BND1501 LLM FORMAT: '1'
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'OK'
```

The output generated by this example program corresponds to the output generated in example 1 with the exception of the message CCM0998 after program termination.

7.6 Example 6 - Assembler application routine

This example uses the interfaces IEDTGTM, IEDTPARL and IEDTEXE to read marked file names from the current work file and import the associated files in sequence into free work areas.

```
           TITLE 'BEISPIEL6'
*********************************************************************
*                                                                   *
* Example 6                                                         *
*                                                                   *
* This example implements a user routine which makes it             *
* possible to read in multiple files which have been marked         *
* in a list of file names.                                          *
* It uses the iedtgtm interface to read all the marked records      *
* from the current work area.                                       *
* It is expected that the records contain file names which, for     *
* example, were created with SHOW F=* TO 1.                         *
*                                                                   *
* The example program performs the following actions:               *
*                                                                   *
* 1) Execute a loop in which all the marked records in the          *
*    work file are read.                                            *
* 2) For each marked file, iedtparl is called to search for         *
*    a free work file.                                              *
* 3) If a free work file is available, the file is read in using    *
*    @COPY FILE= (via iedtexe)                                      *
*                                                                   *
*********************************************************************
*
CMULTI   CSECT
CMULTI   AMODE ANY
CMULTI   RMODE ANY
         GPARMOD 31
*
         STM   R14,R12,12(R13)    * SAVE REGISTERS
         LR    R10,R15            * ENTER VALUES IN BASE REGISTER
         USING CMULTI,R10
*
*        ENTER VALUES IN PL USING THE SUPPLIED GLCB
*
         L     R11,0(,R1)         GLCB FROM EDT
         ST    R11,EXEPL          -> TO THE EXE PL
         ST    R11,PARLPL         -> TO THE PARL PL
         ST    R11,GTMPL          -> TO THE GTM PL
         USING EDTGLCB,R11
*
```

```
*        SUPPLY SAVE AREA FOR UP CALLS
*
         LR    R7,R13             SAVE R13
         LA    R13,SAVEAREA
*
*        SUPPLY AMCB  FOR GTM
*
         MVC   EAMFILE,EGLFILE    CURRENT WORK FILE FROM GLCB
         MVC   EAMDISP,=A(1)      READ RECORD BY KEY
         MVC   EAMLKEY1,=Y(8)     ENTER VALUES IN LENGTH FIELDS
         MVC   EAMPKEY,=Y(8)
         MVC   EAMPREC,=Y(54)
*
*        SUPPLY AMCB  FOR PARL
*
         MVC   PAMFILE(3),=C'L00' WORK FILE FOR PARL
         MVC   PAMLKEY1,=Y(8)     ENTER VALUES IN LENGTH FIELDS
         MVC   PAMPKEY,=Y(8)
         MVC   PAMPREC,=Y(EPLPARLL)
*
*        LOOP THROUGH ALL MARKED RECORDS
*
         LA    R3,0(0,0)          COUNTER FOR WORK FILES
LOOP     DS    0Y
         LA    R1,GTMPL           ADDRESS OF EDT STATEMENT
         L     R15,=V(IEDTGTM)    GTM ROUTINE
         BALR  R14,R15
         CLC   EGLMRET,=Y(EAMRETOK) GTM OK ?
         LA    R1,ERRGTM          ERROR MESSAGE
         BNE   LOOPERR            ERROR OUTPUT
         CLI   EGLSR1,EAMOK12     LAST MARKED RECORD?
         BE    LOOPEX             NORMAL OUTPUT
*
*        SEARCH FOR FREE WORK FILE
*
LOOPI    DS    0Y
         LA    R4,ARBDATNR        PRINTABLE NUMBERS
         LA    R4,0(R3,R4)        ADD COUNTERS
         MVC   PAMFILE+1(2),0(R4) TRANSFER NUM TO AMCB
         LA    R1,PARLPL          FOR PARL CALL
         L     R15,=V(IEDTGET)    GET ROUTINE (PARL)
         BALR  R14,R15
         CLC   EGLMRET,=Y(EAMRETOK)   PARL OK?
         LA    R1,ERRPARL
         BNE   LOOPERR            ERROR OUTPUT
         CLI   EPLEMPTY,'1'       WORK FILE EMPTY?
         BE    LOOPIEX            EXIT LOOP
```

```
         LA    R3,2(,R3)          NEXT WORK FILE IN 2ND STEP BECAUSE
*                                 2-DIGIT IN WORK FILE NO.
         CH    R3,=Y(44)          LAST WORK FILE REACHED?
         BH    LOOPEX             NORMAL OUTPUT
         B     LOOPI              TRY NEXT
*
LOOPIEX  DS    0Y
         MVC   ADAT,PAMFILE+1     WORK FILE IN SETF STATEMENT
         MVI   FILE,' '           PRELIMINARY DELETE OF FILE NAME
         MVC   FILE+1(53),FILE
         LH    R15,EAMLREC        LENGTH OF READ RECORD
         BCTR  R15,0              MINUS 1 FOR EX
         EX    R15,EXMVCFIL       FILE NAME IN @COPY STATEMENT
         LA    R1,EXEPL           FOR EXE CALL
         L     R15,=V(IEDTEXE)
         BALR  R14,R15
         CLC   EGLMRET,=Y(EUPRETOK)   EXE OK?
         LA    R1,ERREXE
         BNE   LOOPERR            ERROR OUTPUT
         MVC   KEY1(8),KEY        NEW BASIS IS READ RECORD --- (1)
         B     LOOP
*
LOOPEX   DS    0Y
         MVC   EGLMRET,=Y(EUPRETOK)   RC OK
         MVI   EGLSR1,EUPOK00
         LR    R13,R7                 RESTORE R13
         LM    R14,R12,12(R13)
         BR    R14                    RETURN TO EDT
*
LOOPERR  DS    0Y                              ---------------------- (2)
         MVC   EGLMRET,=Y(EUPRTERR)   ERROR IN USER PROGRAM
         MVI   EGLSR1,EUPOK00
         MVC   EGLRMSG(ERRMSGL+2),0(R1)
         LR    R13,R7                 RESTORE R13
         LM    R14,R12,12(R13)
         BR    R14
*********************************************************************
*        INITIALIZATION ROUTINE                                     *
*********************************************************************
         ENTRY CMULTI@I
CMULTI@I DS    0D                     ------------------------------- (3)
         STM   R14,R12,12(R13)
         USING CMULTI@I,R15
         L     R11,0(,R1)             ADDRESS OF GLCB
         USING EDTGLCB,R11
         MVC   EGLCCSN,=C'EDF041  '
         LM    R14,R12,12(R13)
         BR    R14
```

```
         DROP  R11,R15
         EJECT
*************************************************************************
*                                                                       *
*        CONSTANTS                                                      *
*                                                                       *
*************************************************************************
*
*        REGISTER DEFINITIONS
*
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         EJECT
*************************************************************************
*                                                                       *
*        FIELDS                                                         *
*                                                                       *
*************************************************************************
SAVEAREA DS    18F                    SAVE AREA
*
EXMVCFIL MVC   FILE(0),REC            FILE NAME IN COPY STATEMENT
*
REC      DS    CL54                   AREA FOR FILE NAMES
*
ERRGTM   DC    Y(ERRMSGL)
         DC    'FEHLER BEI IEDTGTM   '
ERRMSGL  EQU   *-ERRGTM-2
ERRPARL  DC    Y(ERRMSGL)
         DC    'FEHLER BEI IEDTPARL  '
ERREXE   DC    Y(ERRMSGL)
         DC    'FEHLER BEI IEDTEXE   '
```

```
*
CMD1     DC    Y(CMD1END-CMD1)         EDT STATEMENT: COPY
         DS    CL2
         DC    C'@SETF('
ADAT     DC    C'00'
         DC    ');@COPY FILE='
FILE     DC    CL54' '
CMD1END  EQU   *
*
ARBDATNR DC    C'000102030405060708091012131415161718192021 22'
*
*        PARAMETER LIST FOR THE EDT EXE INTERFACE
*
EXEPL    DC    A(0)                    EDTGLCB ADDRESS
         DC    A(CMD1)                 ADDRESS OF STATEMENT
*
*        PARAMETER LIST FOR THE EDT GTM INTERFACE
*
GTMPL    DC    A(0)                    EDTGLCB ADDRESS
         DC    A(EDTAMCB)              EDTAMCB ADDRESS
         DC    A(KEY1)                 KEY1 (IN) ADDRESS
         DC    A(KEY)                  KEY (OUT) ADDRESS
         DC    A(REC)                  ADDRESS OF READ RECORD
*
*        PARAMETER LIST FOR THE EDT PARL INTERFACE
*
PARLPL   DC    A(0)                    EDTGLCB ADDRESS
         DC    A(PEDTAMCB)             EDTAMCB ADDRESS
         DC    A(KEY1)                 KEY1 (IN) ADDRESS
         DC    A(KEY)                  KEY (OUT) ADDRESS
         DC    A(EDTPARL)              ADDRESS OF INFO OUTPUT
*
KEY1     DC    2A(0)                   KEY FOR GTM
KEY      DC    2A(0)                   KEY (RETURN VALUE)
*
*        EDT-SPECIFIC INTERFACE MACROS IN V17.0A
*
         IEDTAMCB C,VERSION=2
*
         IEDTAMCB C,P,VERSION=2
*
         IEDTPARL C,VERSION=4
*
         IEDTGLCB D,VERSION=2
CMULTI   CSECT
         END
```

Explanations

(1) `EDTGTM` reads starting from the specified key in the direction specified by `EAMDISP` (here forwards). As a result, the new key must become the new starting point.

(2) For reasons of clarity, there is only a brief indication of the error handling mechanisms. However, at the very least, the original return code provided by the interface should be prepared and output.

(3) Specifying the initialization routine also causes `CMULTI` to be called with the V17 `GLCB`.

If the procedure explained in "Producing user routines in Assembler" is stored in a file named ASSMOD.DO in BS2000 and the source file
is stored as the S element ANWEND2.ASS in the library EDT.BEISPIELE then the above program can be compiled and linked with

```
/CALL-PROC ASSMOD.DO,(2)
```

The generated program can then be loaded from EDT with

@USE COMMAND='*',ELEMENT=CMULTI,MODLIB=EDT.BEISPIELE

The procedure ASSMOD.DO generates output resembling the following:

```
%  BLS0523 ELEMENT 'ASSEMBH', VERSION '012', TYPE 'C' FROM LIBRARY ':MARS:
$TSOS.SYSPRG.ASSEMBH.012' IN PROCESS
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '01.2C00' OF '2002-03-06' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 2002. ALL RIGHTS
RESERVED
%  ASS6010 V01.2C00 OF BS2000 ASSEMBH  READY
%  ASS6011 ASSEMBLY TIME: 480 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 12 MSEC
%  ASS6012 END OF ASSEMBH
```

The operation of the routine will be demonstrated in L mode, i.e. EDT must already be loaded and be waiting for input in L mode:

```
  1.     @SHOW F=* TO 1                          ----------- (1)
550.     @O&F'BEISPIEL'                          ----------- (2)
550.     @USE COM='*',E=CMULTI,M=EDT.BEISPIELE   ----------- (3)
550.     *CMULTI                                 ----------- (4)
  %  EDT5999 FEHLER BEI IEDTEXE                  ----------- (5)
  1. @PROC 22
  1. @STA=PAR TO 1                              ----------- (6)
254. @ON & P '$ ='
  7.0000     % =    1.0000  $ =  549.0000  * =  550.0000  ? =   22.0000
 18.0000     % =    1.0000  $ =  309.0000  * =  310.0000  ? =    0.0000
 29.0000     % =    1.0000  $ =  187.0000  * =  188.0000  ? =    0.0000
 40.0000     % =    1.0000  $ =  179.0000  * =  180.0000  ? =    0.0000
 51.0000     % =    1.0000  $ =  235.0000  * =  236.0000  ? =    0.0000
 62.0000     % =    1.0000  $ =    1.0000  * =    1.0000  ? =    0.0000
....
```

Explanations

(1) The file list is constructed in the work file. Here, for example, the list has 549 entries.

(2) All files which have the name component BEISPIEL are searched for and marked.

(3) The user routine CMULTI is loaded from the library EDT.BEISPIELE and is used to process user statements which start with '*' .

(4) The user routine is called. Since it does not evaluate the input string, it could be called with any other statement, e.g. *XXX.

(5) The message comes from the user routine since IEDTEXE has supplied a return code. The library EDT.BEISPIELE is also found. This cannot be read using @COPY.

(6) @STA=PAR provides a view of file occupancy and makes it clear that files have been read into work files 1 to 4.

8 Appendix - C header files

This Appendix presents the layout of the C header files. For comments on their use, see the preceding sections. The equivalent Assembler macros are described in the section “Generation and structure of the control blocks”.

8.1 Include file for programming in C

To make it possible to call EDT from a C program, macros, in the form of include files, are supplied for the definition, initialization and modification of the EDT control blocks. The return codes are defined as symbolic constants.

Both formats (V17 format and V16 format) of the interfaces are always generated. In all cases, these can be addressed via `structure-name_v16` or `structure-name_v17` (e.g. `iedglcb_v17 or iedglcb_v16`).

By setting the value `EDT_V17` for `#define`, users can specify whether the standard names of the control blocks and the abbreviated names for initialization or access operations are to correspond to V17 or V16 format. If `EDT_V17` is defined then V17 is used as the default, otherwise the V16 format.

The individual control block fields are described in section “Generation and structure of thecontrol blocks”.

8.1.1 iedtgle.h

Function prototypes for the entry points of the `IEDTGLE` interface.

Note

This include sets an `#include` for all the other include files.
This include is therefore sufficient for the use of the EDT interfaces.

```
/*
*********************************************************************
** function prototypes
*********************************************************************
*/
extern void IEDTINF(iedglcb *glcb);
extern void IEDTCMD(iedglcb *glcb,iedupcb *upcb,iedbuff *cmd,
                    iedbuff *msg1,iedbuff *msg2);
extern void IEDTEXE(iedglcb *glcb,iedbuff *cmd);
extern void IEDTGET(iedglcb *glcb,iedamcb *amcb,iedbyte *key1,
                    iedbyte *key,void *rec_or_parg_or_parl);
extern void IEDTGTM(iedglcb *glcb,iedamcb *amcb,iedbyte *key1,
                    iedbyte *key,iedbyte *rec);
extern void IEDTPUT(iedglcb *glcb,iedamcb *amcb,iedbyte *key,iedbyte *rec);
extern void IEDTPTM(iedglcb *glcb,iedamcb *amcb,iedbyte *key);
extern void IEDTDEL(iedglcb *glcb,iedamcb *amcb,iedbyte *key1,iedbyte *key2);
extern void IEDTREN(iedglcb *glcbiedamcb *amcb,iedbyte *key1,iedbyte *key2);
```

8.1.2 iedglcb.h

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

```
/*
*********************************************************************
** common typedefs
*********************************************************************
*/
#ifndef IEDT_TYPES
typedef unsigned char iedbyte;
typedef unsigned short iedshort;
typedef unsigned long iedlong;
typedef unsigned short iedutf16;
#define IEDT_TYPES
#endif
/*
*********************************************************************
** IEDGLCB parameter block V16
*********************************************************************
*/
typedef struct IEDGLCB_v16_mdl {
    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 1  */
    /* returncode structure */
    union {
        struct {
            struct {
                iedbyte subcode2;
                iedbyte subcode1;
            } subcode;
            union {
                iedshort maincode;
                struct {
                    iedbyte maincode2;
                    iedbyte maincode1;
```

```
            } main_returncode;
        } mc;
    } structured_rc;
    iedlong rc_nbr;              /* general return code */
} rc;
/* info size or displacement of invalid command */
union {
    iedlong memo_size;           /* information of memory size */
    iedlong displ_to_cmd;        /* displacement of invalid command */
} size_or_displacement;
/* return message field */
union {
    struct {
        iedshort rmsgl;          /* message length */
        iedbyte rmsgf[80];       /* message field */
    } structured_msg;
    iedbyte rmsg[82];            /* return message */
} return_message;
/* code of sending key */
iedbyte key_code;
/* indicator byte */
struct {
    iedbyte not_used_1:1;        /* not used */
    iedbyte not_used_2:1;        /* not used */
    iedbyte reorg_allowed:1;     /* reorganisation allowed */
    iedbyte not_used_3:1;        /* not used */
    iedbyte stxit_allowed:1;     /* EDT STXIT allowed */
    iedbyte data_initiated:1;    /* EDT data initiated */
    iedbyte data_add_valid:1;    /* EDT data addr. valid */
    iedbyte 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 */
iedbyte filename[8];
```

```
    /* user parameter 1 */
    union {
        iedbyte user_param1_char[4];
        void *user_param1_pointer;
    } user_param1;
    /* user parameter 2 */
    union {
        iedbyte user_param2_char[4];
        void *user_param2_pointer;
    } user_param2;
    /* user parameter 3 */
    union {
        iedbyte user_param3_char[4];
        void *user_param3_pointer;
    } user_param3;
} iedglcb_v16;
/*
*********************************************************************
** IEDGLCB parameter block V17
*********************************************************************
*/
typedef struct IEDGLCB_v17_mdl {
    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 2  */
    /* returncode structure */
    union {
        struct {
            struct {
                iedbyte subcode2;
                iedbyte subcode1;
            } subcode;
            union {
                iedshort maincode;
                struct {
                    iedbyte maincode2;
                    iedbyte maincode1;
                } main_returncode;
            } mc;
```

```
    } structured_rc;
    iedlong rc_nbr;           /* general return code */
} rc;
/* info size or displacement of invalid command */
union {
    iedlong memo_size;        /* information of memory size */
    iedlong displ_to_cmd;     /* displacement of invalid command */
} size_or_displacement;
/* return message field */
union {
    struct {
        iedshort rmsgl;            /* message length */
        iedbyte rmsgf[80];         /* message field */
    } structured_msg;
    iedbyte rmsg[82];              /* return message */
} return_message;
/* code of sending key */
iedbyte key_code;
/* indicator byte */
struct {
    iedbyte compatible_format:1;   /* compatible/extended format */
    iedbyte ilcs_environment:1;    /* ILCS environment */
    iedbyte not_used_1:1;          /* not used */
    iedbyte not_used_2:1;          /* not used */
    iedbyte stxit_allowed:1;       /* EDT STXIT allowed */
    iedbyte data_initiated:1;      /* EDT data initiated */
    iedbyte data_add_valid:1;      /* EDT data addr. valid */
    iedbyte 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 */
iedbyte filename[8];
/* user parameter 1 */
union {
    iedbyte user_param1_char[4];
    void *user_param1_pointer;
} user_param1;
```

```
    /* user parameter 2 */
    union {
        iedbyte user_param2_char[4];
        void *user_param2_pointer;
    } user_param2;
    /* user parameter 3 */
    union {
        iedbyte user_param3_char[4];
        void *user_param3_pointer;
    } user_param3;
    /* coded character set name for sub programm communication */
    iedbyte ccsn[8];
    /* indicator byte */
    struct {
        iedbyte comp_mode_running:1;   /* compatible/extended format */
        iedbyte not_used_3:1;          /* not used */
        iedbyte not_used_4:1;          /* not used */
        iedbyte not_used_5:1;          /* not used */
        iedbyte not_used_6:1;          /* not used */
        iedbyte not_used_7:1;          /* not used */
        iedbyte not_used_8:1;          /* not used */
        iedbyte not_used_9:1;          /* not used */
    } indicator2;
    iedbyte reserve[3];                /* reserved */
} iedglcb_v17;
/*
********************************************************************
** IEDGLCB parameter block default
********************************************************************
*/
#ifdef EDT_V17
typedef iedglcb_v17 iedglcb;
#define IEDGLCB_mdl IEDGLCB_v17_mdl
#else
typedef iedglcb_v16 iedglcb;
#define IEDGLCB_mdl IEDGLCB_v16_mdl
#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        */
#define IEDGLCBcmd_compatibility    48 /* V17: compatibility violation */
/* 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   */
#define IEDGLCBccsn_error           20 /* V17: error in ccsn parameter */
#define IEDGLCBconversion_error     24 /* V17: conversion error        */
/* 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     */
#define IEDGLCBacc_ccsn_error       40 /* V17: error in ccsn parameter */
#define IEDGLCBacc_conversion_error 44 /* V17: conversion error        */
```

```
/*
*********************************************************************
** 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 */
/*
*********************************************************************
** macros for initialization, access, and modification
*********************************************************************
*/
#define IEDGLCB_UNIT_66    66
#define IEDGLCB_FUNCT_0     0
#define IEDGLCB_VERS_1      1
#define IEDGLCB_VERS_2      2
#define IEDGLCB_INIT_V16 \
    { 66,0,1,{0},0,{0},0,{0},0,0,"        ",{0},{"    "},{"    "} }
#define IEDGLCB_INIT_V17 \
    { 66,0,2,{0},0,{0},0,{0,1,0,0,0,0,0,0},0,0,"        ",{0},{"    "},
{"    "},"        ",{0,0,0,0,0,0,0,0},{0,0,0} }
#ifdef EDT_V17
#define IEDGLCB_INIT        IEDGLCB_INIT_V17
#define IEDGLCB_VERS_STD    IEDGLCB_VERS_2
#else
#define IEDGLCB_INIT        IEDGLCB_INIT_V16
#define IEDGLCB_VERS_STD    IEDGLCB_VERS_1
#endif
#define IEDGLCB_MOD_VERS(p,v)   (p).version = v
#define IEDGLCB_MOD_IFID(p,u,f,v) \
    (p).unit  = u, (p).function = f, (p).version  = v
#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    rc.structured_rc.mc.main_returncode.maincode1
#define IEDGLCB_RC_NBR          rc.rc_nbr
#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_RC_NIL    -1
#define IEDGLCB_RC_NULL    0
#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
#define IEDGLCB_MSG    return_message.structured_msg.rmsgf
#define IEDGLCB_MSGL   return_message.structured_msg.rmsgl
/*
*********************************************************************
** layout of buffers (command, message)
*********************************************************************
*/
typedef struct IEDBUFF_mdl {
    iedshort length;        /* length including all fields */
    iedshort unused;        /* unused field */
    iedbyte text[1];        /* up to 256 (V16) or 32768 (V17) bytes */
} iedbuff;
```

8.1.3 iedupcb.h

Definitions and macros for the subroutine control block EDTUPCB:

```
/*
*********************************************************************
** common typedefs
*********************************************************************
*/
#ifndef IEDT_TYPES
typedef unsigned char iedbyte;
typedef unsigned short iedshort;
typedef unsigned long iedlong;
typedef unsigned short iedutf16;
#define IEDT_TYPES
#endif
/*
*********************************************************************
** IEDUPCB parameter block V16
*********************************************************************
*/
typedef struct IEDUPCB_v16_mdl {
    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 2  */
    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;
    /* inhibit flag byte */
    union {
        struct {
            iedbyte not_used_1:1;        /* reserved */
            iedbyte no_text_at_exit:1;   /* @HALT/@RET <text> */
            iedbyte no_edit_only:1;      /* @EDIT ONLY */
            iedbyte no_edit:1;           /* @EDIT */
            iedbyte no_user_prog:1;      /* @RUN, @USE */
            iedbyte no_bkpt:1;           /* @SYSTEM */
            iedbyte no_cmd:1;            /* @SYSTEM <string> */
```

```
            iedbyte no_exec:1;           /* @EXEC/@LOAD */
        } bit;
        iedbyte byte;
    } inhibit;
    /* reserve */
    iedbyte reserve[3];
} iedupcb_v16;
/*
*********************************************************************
** IEDUPCB parameter block V17
*********************************************************************
*/
typedef struct IEDUPCB_v17_mdl {
    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 3  */
    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;
    /* inhibit flag byte */
    union {
        struct {
            iedbyte no_mode:1;             /* @MODE */
            iedbyte no_text_at_exit:1;     /* @HALT/@RET <text> */
            iedbyte no_edit_only:1;        /* @EDIT ONLY */
            iedbyte no_edit:1;             /* @EDIT */
            iedbyte no_user_prog:1;        /* @RUN, @USE */
            iedbyte no_bkpt:1;             /* @SYSTEM */
            iedbyte no_cmd:1;              /* @SYSTEM <string> */
            iedbyte no_exec:1;             /* @EXEC/@LOAD */
        } bit;
        iedbyte byte;
    } inhibit;
    /* reserve */
    iedbyte reserve[3];
} iedupcb_v17;
```

```
/*
*********************************************************************
** IEDUPCB parameter block default
*********************************************************************
*/
#ifdef EDT_V17
typedef iedupcb_v17 iedupcb;
#define IEDUPCB_mdl IEDUPCB_v17_mdl
#else
typedef iedupcb_v16 iedupcb;
#define IEDUPCB_mdl IEDUPCB_v16_mdl
#endif
/*
*********************************************************************
** macros for initialization, access, and modification
*********************************************************************
*/
#define IEDUPCB_UNIT_66   66
#define IEDUPCB_FUNCT_0   0
#define IEDUPCB_VERS_2    2
#define IEDUPCB_VERS_3    3
#define IEDUPCB_INIT_V16 { 66,0,2 }
#define IEDUPCB_INIT_V17 { 66,0,3 }
#ifdef EDT_V17
#define IEDUPCB_INIT       IEDUPCB_INIT_V17
#define IEDUPCB_VERS_STD   IEDUPCB_VERS_3
#else
#define IEDUPCB_INIT       IEDUPCB_INIT_V16
#define IEDUPCB_VERS_STD   IEDUPCB_VERS_2
#endif
#define IEDUPCB_MOD_VERS(p,v)   (p).version = v
#define IEDUPCB_MOD_IFID(p,u,f,v) \
    (p).unit  = u, (p).function = f, (p).version  = v
#define IEDUPCB_SET_NO_INHIBIT(p)   (p).inhibit.byte = 0
```

8.1.4 iedamcb.h

Definitions and macros for the record access control block `EDTAMCB`:

```
/*
*********************************************************************
** common typedefs
*********************************************************************
*/
#ifndef IEDT_TYPES
typedef unsigned char iedbyte;
typedef unsigned short iedshort;
typedef unsigned long iedlong;
typedef unsigned short iedutf16;
#define IEDT_TYPES
#endif
/*
*********************************************************************
** IEDAMCB parameter block V16
*********************************************************************
*/
typedef struct IEDAMCB_v16_mdl {
    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 1  */
    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;
    /* transfer mode flag byte */
    union {
        struct {
            iedbyte not_used_1:5;        /* not used */
            iedbyte locate:1;            /* locate mode */
            iedbyte not_used_2:2;        /* not used */
        } mode_bits;
        iedbyte mode_byte;               /* mode byte */
    } mode_flag;
```

```
/* flag byte */
union {
    struct {
        iedbyte not_used:6;          /* not used */
        iedbyte inh_set_modify:1;    /* inhibit setting modify flag */
        iedbyte ign_mark13:1;        /* ignore mark 13 */
    } flag_bits;
    iedbyte flag_byte;
} flag;
/* reserve */
iedbyte reserve2[2];
/* input parameters */
iedbyte filename[8];         /* workfile */
long displacement;           /* displacement */
iedshort length_key1;        /* length of key1 */
iedshort length_key2;        /* length of key2 */
/* input parameters (only in move mode) */
iedshort length_key_outbuffer;   /* length of key output buffer */
iedshort length_rec_outbuffer;   /* length of rec output buffer */
/* input/output parameters */
iedshort length_key;             /* length of key */
iedshort length_rec;             /* length of record */
/* marks */
union {
    iedshort mark_field;
    struct {
        union {
            iedbyte mark2;               /* upper marks */
            struct {
                iedbyte mark_15:1;       /* mark 15 */
                iedbyte mark_14:1;       /* mark 14 */
                iedbyte mark_13:1;       /* mark 13 */
                iedbyte nouse_1:1;       /* not used */
                iedbyte nouse_2:1;       /* not used */
                iedbyte nouse_3:1;       /* not used */
                iedbyte mark_9:1;        /* mark 9 */
                iedbyte mark_8:1;        /* mark 8 */
            } mark2_bits;
        } upper_marks;
        union {
            iedbyte mark1;               /* lower marks */
            struct {
                iedbyte mark_7:1;        /* mark 7 */
```

```
                    iedbyte mark_6:1;        /* mark 6 */
                    iedbyte mark_5:1;        /* mark 5 */
                    iedbyte mark_4:1;        /* mark 4 */
                    iedbyte mark_3:1;        /* mark 3 */
                    iedbyte mark_2:1;        /* mark 2 */
                    iedbyte mark_1:1;        /* mark 1 */
                    iedbyte nouse_4:1;       /* not used */
                } mark1_bits;
            } lower_marks;
        } mark_bytes;
    } marks;
    /* reserve */
    iedbyte reserve[2];
} iedamcb_v16;
/*
********************************************************************
** IEDAMCB parameter block V17
********************************************************************
*/
typedef struct IEDAMCB_v17_mdl {
    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 2  */
    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;
    /* not used */
    iedbyte reserve1;
    /* flag byte */
    union {
        struct {
            iedbyte not_used:6;        /* not used */
            iedbyte inh_set_modify:1;  /* inhibit setting modify flag */
            iedbyte ign_mark13:1;      /* ignore mark 13 */
        } flag_bits;
        iedbyte flag_byte;
    } flag;
```

```
/* reserve */
iedbyte reserve2[2];
/* input parameters */
iedbyte filename[8];         /* workfile */
long displacement;           /* displacement */
iedshort length_key1;        /* length of key1 */
iedshort length_key2;        /* length of key2 */
/* input parameters */
iedshort length_key_outbuffer;  /* length of key output buffer */
iedshort length_rec_outbuffer;  /* length of rec output buffer */
/* input/output parameters */
iedshort length_key;         /* length of key */
iedshort length_rec;         /* length of record */
/* marks */
union {
    iedshort mark_field;
    struct {
        union {
            iedbyte mark2;               /* upper marks */
            struct {
                iedbyte mark_15:1;       /* mark 15 */
                iedbyte mark_14:1;       /* mark 14 */
                iedbyte mark_13:1;       /* mark 13 */
                iedbyte nouse_1:1;       /* not used */
                iedbyte nouse_2:1;       /* not used */
                iedbyte nouse_3:1;       /* not used */
                iedbyte mark_9:1;        /* mark 9 */
                iedbyte mark_8:1;        /* mark 8 */
            } mark2_bits;
        } upper_marks;
        union {
            iedbyte mark1;               /* lower marks */
            struct {
                iedbyte mark_7:1;        /* mark 7 */
                iedbyte mark_6:1;        /* mark 6 */
                iedbyte mark_5:1;        /* mark 5 */
                iedbyte mark_4:1;        /* mark 4 */
                iedbyte mark_3:1;        /* mark 3 */
                iedbyte mark_2:1;        /* mark 2 */
                iedbyte mark_1:1;        /* mark 1 */
                iedbyte nouse_4:1;       /* not used */
            } mark1_bits;
        } lower_marks;
    } mark_bytes;
```

```
    } marks;
    /* reserve */
    iedbyte reserve[2];
} iedamcb_v17;
/*
*********************************************************************
** IEDAMCB parameter block default
*********************************************************************
*/
#ifdef EDT_V17
typedef iedamcb_v17 iedamcb;
#define IEDAMCB_mdl IEDAMCB_v17_mdl
#else
typedef iedamcb_v16 iedamcb;
#define IEDAMCB_mdl IEDAMCB_v16_mdl
#endif
/*
*********************************************************************
** macros for initialization, access, and modification
*********************************************************************
*/
#define IEDAMCB_UNIT_66   66
#define IEDAMCB_FUNCT_0   0
#define IEDAMCB_VERS_1    1
#define IEDAMCB_VERS_2    2
#define IEDAMCB_INIT_V16 { 66,0,1,0,{0},{0},{0},"        ",0,8,8,8,0,8, }
#define IEDAMCB_INIT_V17 { 66,0,2,0,0,{0},{0},"        ",0,8,8,8,0,8, }
#ifdef EDT_V17
#define IEDAMCB_INIT       IEDAMCB_INIT_V17
#define IEDAMCB_VERS_STD   IEDAMCB_VERS_2
#else
#define IEDAMCB_INIT       IEDAMCB_INIT_V16
#define IEDAMCB_VERS_STD   IEDAMCB_VERS_1
#endif
```

```
#define IEDAMCB_MOD_VERS(p,v)   (p).version = v
#define IEDAMCB_MOD_IFID(p,u,f,v) \
    (p).unit  = u, (p).function = f, (p).version  = v
#define IEDAMCB_SET_NO_MARKS(p)   (p).marks.mark_field = 0
#define MARK_1(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_1
#define MARK_2(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_2
#define MARK_3(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_3
#define MARK_4(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_4
#define MARK_5(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_5
#define MARK_6(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_6
#define MARK_7(p)   (p).marks.mark_bytes.lower_marks.mark1_bits.mark_7
#define MARK_8(p)   (p).marks.mark_bytes.upper_marks.mark2_bits.mark_8
#define MARK_9(p)   (p).marks.mark_bytes.upper_marks.mark2_bits.mark_9
#define MARK_13(p)  (p).marks.mark_bytes.upper_marks.mark2_bits.mark_13
#define MARK_14(p)  (p).marks.mark_bytes.upper_marks.mark2_bits.mark_14
#define MARK_15(p)  (p).marks.mark_bytes.upper_marks.mark2_bits.mark_15
```

8.1.5 iedparg.h

Definitions and macros for the control block EDTPARG (global parameter settings):

```
/*
*********************************************************************
** common typedefs
*********************************************************************
*/
#ifndef IEDT_TYPES
typedef unsigned char iedbyte;
typedef unsigned short iedshort;
typedef unsigned long iedlong;
typedef unsigned short iedutf16;
#define IEDT_TYPES
#endif
/*
*********************************************************************
** IEDPARG parameter block V16
*********************************************************************
*/
typedef struct IEDPARG_v16_mdl {
    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;                  /* function unit number : 66 */
    iedbyte function;               /* function number      : 0  */
    iedbyte version;                /* interface version    : 1  */
    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;
    /* output fields */
    iedbyte EDT_mode;               /* edt modus */
    iedbyte command_symbol;         /* actual '@' */
    iedshort size_window1;          /* size of window 1 */
    iedshort size_window2;          /* size of window 2 */
    iedbyte file_in_window1[8];     /* workfile in window 1 */
    iedbyte file_in_window2[8];     /* workfile in window 2 */
    iedbyte ccs_name[8];            /* global coded character set */
} iedparg_v16;
```

```
/*
*********************************************************************
** IEDPARG parameter block V17
*********************************************************************
*/
typedef struct IEDPARG_v17_mdl {
    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;               /* function unit number : 66 */
    iedbyte function;            /* function number      : 0  */
    iedbyte version;             /* interface version    : 2  */
    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;
    /* output fields */
    iedbyte EDT_mode;            /* edt modus */
    iedbyte command_symbol;      /* actual '@' */
    iedshort size_window1;       /* size of window 1 */
    iedshort size_window2;       /* size of window 2 */
    iedbyte file_in_window1[8];  /* workfile in window 1 */
    iedbyte file_in_window2[8];  /* workfile in window 2 */
    /* reserve */
    iedbyte reserve[10];
} iedparg_v17;
/*
*********************************************************************
** IEDPARG parameter block default
*********************************************************************
*/
#ifdef EDT_V17
typedef iedparg_v17 iedparg;
#define IEDPARG_mdl IEDPARG_v17_mdl
#else
typedef iedparg_v16 iedparg;
#define IEDPARG_mdl IEDPARG_v16_mdl
#endif
```

```
/*
*********************************************************************
** special values in EDT_mode
*********************************************************************
*/
#define IEDPARGmode_fullscreen  0xC6        /* 'F' full screen mode */
#define IEDPARGmode_line        0xD3        /* 'L' line mode */
#define IEDPARGmode_control     0xC3        /* 'C' user control */
/*
*********************************************************************
** macros for initialization, access, and modification
*********************************************************************
*/
#define IEDPARG_UNIT_66    66
#define IEDPARG_FUNCT_0     0
#define IEDPARG_VERS_1      1
#define IEDPARG_VERS_2      2
#define IEDPARG_INIT_V16 \
    { 66,0,1,0,0,0,0,0,"        ","        ","        " }
#define IEDPARG_INIT_V17 \
    { 66,0,2,0,0,0,0,0,"        ","        " }
#ifdef EDT_V17
#define IEDPARG_INIT       IEDPARG_INIT_V17
#define IEDPARG_VERS_STD   IEDPARG_VERS_2
#else
#define IEDPARG_INIT       IEDPARG_INIT_V16
#define IEDPARG_VERS_STD   IEDPARG_VERS_1
#endif
#define IEDPARG_MOD_VERS(p,v)    (p).version = v
#define IEDPARG_MOD_IFID(p,u,f,v) \
    (p).unit  = u, (p).function = f, (p).version  = v
```

8.1.6 iedparl.h

Definitions and macros for the control block EDTPARL (work file-specific parameter settings):

```
/*
*********************************************************************
** common typedefs
*********************************************************************
*/
#ifndef IEDT_TYPES
typedef unsigned char iedbyte;
typedef unsigned short iedshort;
typedef unsigned long iedlong;
typedef unsigned short iedutf16;
#define IEDT_TYPES
#endif
/*
*********************************************************************
** IEDPARL parameter block V16
*********************************************************************
*/
typedef struct IEDPARL_v16_mdl {
    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;        /* function unit number : 66 */
    iedbyte function;     /* function number      : 0  */
    iedbyte version;      /* interface version    : 3  */
    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;
    /* output fields */
    iedbyte first_line_window[8]; /* number of first line */
    /* in window */
    iedshort first_col_window;    /* first column in window */
    iedshort record_length_max;   /* max. record length in */
```

```
    /* in fullscreen mode */
    iedbyte par_inf;          /* INF on/off (1/0) */
    iedbyte par_low;          /* LOWER on/off (1/0) */
    iedbyte par_hex;          /* HEX on/off (1/0) */
    iedbyte par_edit_long;    /* EDIT-LONG on/off (1/0) */
    iedbyte par_scale;        /* SCALE on/off (1/0) */
    iedbyte par_protection;   /* PROTECTION on/off (1/0) */
    iedbyte structure_symbol; /* structure symbol */
    iedbyte open_flag;        /* open flag (I/P/R/S/X/O) */
    iedbyte empty_flag;       /* empty y/n (1/0) */
    iedbyte modified_flag;    /* modified y/n (1/0) */
    iedbyte std_file[54];     /* standard file name */
    iedbyte std_library[54];  /* standard library name */
    iedbyte std_plam_type[8]; /* standard plam type */
    iedbyte std_code;         /* standard code (E/I) */
    iedbyte not_used1[3];     /* reserved */
    iedbyte first_line1[8];   /* number of first line in window 1 */
    iedshort first_col1;      /* first column in window 1 */
    iedbyte first_line2[8];   /* number of first line in window 2 */
    iedshort first_col2;      /* first column in window 2 */
    iedbyte index_window1;    /* INDEX OFF/ON/FULL (0/1/2) window 1 */
    iedbyte index_window2;    /* INDEX OFF/ON/FULL (0/1/2) window 2 */
    /* description of opened data file */
    union {
        iedbyte common_area[260];     /* common area */
        struct {
            iedbyte file_name[54];    /* name of opened file or lib */
            iedbyte plam_elem[64];    /* name of plam element */
            iedbyte plam_vers[24];    /* name of plam version */
            iedbyte plam_type[8];     /* plam type */
        } file_or_plam_elem;
        struct {
            iedbyte ufs_name[256];    /* name of opened ufs file */
            iedbyte code;             /* code of opened ufs file (?)*/
        } ufs_file;
    } file_description;
    /* reserved */
    iedbyte not_used2[8];
} iedparl_v16;
```

```
/*
*********************************************************************
** IEDPARL parameter block V17
*********************************************************************
*/
typedef struct IEDPARL_v17_mdl {
    /* interface identifier structure */
#pragma aligned 4
    iedshort unit;            /* function unit number : 66 */
    iedbyte function;         /* function number      : 0  */
    iedbyte version;          /* interface version    : 4  */
    /* returncode unused, will be returned in control block IEDGLCB */
    iedlong rc_nbr;
    /* output fields */
    iedbyte first_line_window[8]; /* number of first line */
    /* in window */
    iedshort first_col_window;  /* first column in window */
    iedshort record_length_max; /* max. record length in */
    /* in fullscreen mode */
    iedbyte par_inf;          /* INF on/off (1/0) */
    iedbyte par_low;          /* LOWER on/off (1/0) */
    iedbyte par_hex;          /* HEX on/off (1/0) */
    iedbyte par_edit_long;    /* EDIT-LONG on/off (1/0) */
    iedbyte par_scale;        /* SCALE on/off (1/0) */
    iedbyte par_protection;   /* PROTECTION on/off (1/0) */
    iedbyte structure_symbol; /* structure symbol (if EBCDIC) */
    iedbyte open_flag;        /* open flag (I/P/R/S/X/O) */
    iedbyte empty_flag;       /* empty y/n (1/0) */
    iedbyte modified_flag;    /* modified y/n (1/0) */
    iedbyte std_file[54];     /* standard file name */
    iedbyte std_library[54];  /* standard library name */
    iedbyte std_plam_type[8]; /* standard plam type */
    iedbyte not_used1[4];     /* reserved */
    iedbyte first_line1[8];   /* number of first line in window 1 */
    iedshort first_col1;      /* first column in window 1 */
    iedbyte first_line2[8];   /* number of first line in window 2 */
    iedshort first_col2;      /* first column in window 2 */
    iedbyte index_window1;    /* INDEX OFF/ON/FULL (0/1/2) window 1 */
    iedbyte index_window2;    /* INDEX OFF/ON/FULL (0/1/2) window 2 */
```

```
    /* description of opened data file */
    union {
        iedbyte common_area[1024];  /* common area */
        struct {
            iedbyte file_name[54];  /* name of opened file or lib */
            iedbyte plam_elem[64];  /* name of plam element */
            iedbyte plam_vers[24];  /* name of plam version */
            iedbyte plam_type[8];   /* plam type */
        } file_or_plam_elem;
        struct {
            iedbyte ufs_name[1024]; /* name of opened ufs file */
        } ufs_file;
    } file_description;
    /* charset information */
    iedbyte ccsn[8];                /* coded character set name */
    iedbyte ccsn_global;            /* ccsn is global (1/0) */
    /* TODO: other local par settings? */
    iedutf16 sym_structure;         /* structure symbol */
} iedparl_v17;
/*
*********************************************************************
** IEDPARL parameter block default
*********************************************************************
*/
#ifdef EDT_V17
typedef iedparl_v17 iedparl;
#define IEDPARL_mdl IEDPARL_v17_mdl
#else
typedef iedparl_v16 iedparl;
#define IEDPARL_mdl IEDPARL_v16_mdl
#endif
```

```
/*
*********************************************************************
** special values in open_flag
*********************************************************************
*/
#define IEDPARLopen_isam    0xC9 /* 'I' ISAM file virtually opened */
#define IEDPARLopen_plam    0xD7 /* 'P' PLAM element opened */
#define IEDPARLopen_real    0xD9 /* 'R' ISAM file really opened */
#define IEDPARLopen_sam     0xE2 /* 'S' SAM file virtually opened */
#define IEDPARLopen_ufs     0xE7 /* 'X' UFS file virtually opened */
#define IEDPARLopen_no      0xD6 /* 'O' no file opened */
/*
*********************************************************************
** special on/off values
*********************************************************************
*/
#define IEDPARLoff           0xF0 /* '0' OFF */
#define IEDPARLon            0xF1 /* '1' ON */
/*
*********************************************************************
** special values in index
*********************************************************************
*/
#define IEDPARLindex_off     0xF0 /* '0' INDEX OFF */
#define IEDPARLindex_on      0xF1 /* '1' INDEX ON */
#define IEDPARLindex_full    0xF2 /* '2' EDIT FULL */
/*
*********************************************************************
** macros for initialization, access, and modification
*********************************************************************
*/
#define IEDPARL_UNIT_66    66
#define IEDPARL_FUNCT_0     0
#define IEDPARL_VERS_3      3
#define IEDPARL_VERS_4      4
#define IEDPARL_INIT_V16 { 66,0,3 }
#define IEDPARL_INIT_V17 { 66,0,4 }
#ifdef EDT_V17
```

```
#define IEDPARL_INIT        IEDPARL_INIT_V17
#define IEDPARL_VERS_STD    IEDPARL_VERS_4
#else
#define IEDPARL_INIT        IEDPARL_INIT_V16
#define IEDPARL_VERS_STD    IEDPARL_VERS_3
#endif
#define IEDPARL_MOD_VERS(p,v)    (p).version = v
#define IEDPARL_MOD_IFID(p,u,f,v) \
    (p).unit  = u, (p).function = f, (p).version  = v
```

9 Glossary

Batch mode

Batch mode is the EDT operating mode in which no terminal is present. EDT can then only operate in L mode.

Character set

EDT V17.0A makes it possible to process texts in all the character sets made available by XHCS. In addition to the character sets supported in EDT V16.6B and compatibility mode, the character sets supported in Unicode mode include the three Unicode character sets, ISO character sets and, if necessary, the 7-bit character sets.
In each work file it is possible to configure a different character set so that texts in different character sets can be processed in parallel. A communications character set is configured to permit communication with a terminal.

Communications character set

Character set used by EDT in Unicode mode to communicate with the terminal. This can be different from the character set used by the current or any other work file. The communications character set is usually optimally suited to the capabilities offered by the terminal.

Compatibility mode

Compatibility mode is an EDT V17.0A operating mode. Compatibility mode provides the full functionality of EDT V16.6B. However, the extended functions of EDT V17.0A cannot be used. For example, it is not possible to process Unicode files and the record length continues to be restricted to 256 bytes.Under some circumstances, there may be an implicit switchover to Unicode mode if a Unicode file is read or an explicit switchover may occur if a @MODE statement is entered.

Data window

Field in the data window in which the current work file is displayed. The work file records are output in the data window's screen lines.

Delimiter characters

Literals are usually enclosed by the delimiter character `apostrophe` (or single quote, default value '). When a search is performed using the @ON statement, it is possible to use a special delimiter character, namely the `quotation mark` (default value "). This specifies that a string is only recognized as a hit if it is delimited by text delimiters. The text delimiter characters consist of a configurable set of characters which, by default, include the space character, parentheses etc.

@DO procedure

A @DO procedure is an EDT procedure which is stored in a work file. It can be run by means of a @DO statement. @DO procedures provide a number of statements used for runtime control. At call time, it is possible to pass
parameters to these statements which can also be nested.

EDT procedures

An EDT procedure is a sequence of entries, statements and/or records sent to EDT and stored in a file (@INPUT procedure) or work file (@DO procedure).

EDT start procedure

The EDT start procedure is a special @INPUT procedure which (if present) is run when EDT is started.

EDT statement symbol

The EDT statement symbol is a special character used to identify statements. By default, this is the character @ which is therefore consistently used in this document.

EDT variables

EDT variables are containers which can be used to store values across work file boundaries. EDT variables are only valid for the current EDT session.
There are three types of variables which can be assigned the corresponding values. 21 variables of each variable type are available.

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

Full screen mode (F mode)

Full screen mode (F mode) is an EDT work mode. In F mode, the entire screen is available as a work window for the entry of data and statements. It is possible to switch from F mode to L mode. EDT can only operate in interactive mode in F mode.

@INPUT procedure

An @INPUT procedure is an EDT procedure which is stored in a file or library element. It can be run by means of an @INPUT statement. The runtime control statements are not (directly) available in @INPUT procedures. These
statements cannot be nested and it is not possible to pass any parameters. It is, however, possible to call @DO statements.

Interactive mode

Interactive mode is the EDT operating mode in which a terminal is present. This is the only mode in which EDT can operate in F mode.

Line mode (L mode)

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

Line number

A line number is assigned to every record in a work file. This line number uniquely identifies the record. A line number is the current line number in which data is entered in L mode.

Operating mode

Since it was not possible to implement the extensions required for Unicode support in a way which would ensure compatibility, a new EDT operating mode has been introduced. EDT V17.0A can therefore be used in two modes: Unicode mode and compatibility mode.
In Unicode mode, a range of extensions are available. Most importantly, it is possible to process Unicode files (only) in Unicode mode.
However, this mode is not compatible with EDT V16.6B in all respects.
In contrast, compatibility mode provides the full functionality of EDT V16.6B. However, the extensions are not available in this mode.
By default, EDT V17.0A is started in compatibility mode. A new statement is available to start it in Unicode mode.

Operating types

A distinction is made between two types of operation depending on whether or not a terminal is present. In interactive mode, a terminal is present whereas it is not in batch mode.

Record mark

Every record in a work file can be flagged with a record mark which is invisible to users. These marks can be set, queried and deleted using statements and statement codes. Once marked, records can, for example, be copied or deleted.

Screen dialog

The statement @DIALOG – which can only be entered at the subroutine interface or by SYSDTA – is used to switch EDT to screen dialog. In screen dialog, the preceding read operation is interrupted and EDT reads its input from the terminal in F mode (or in L mode after the entry of @EDIT). The screen dialog can be exited again with @HALT, @END, @RETURN or [K1]. EDT then continues the interrupted read operation.

Screen line

Lines in the data window in which the records of the current work file are output.

Statement

Inputs to EDT take the form either of records or statements. Statements are used either to activate EDT functions or make parameter settings. In order to distinguish between statements and records, statements must be entered in line mode with the EDT statement symbol. In F mode, statements are entered in the statement line There are also statement codes which are entered in the statement code column.

Statement buffer

EDT saves the most recent statements entered in F mode in a buffer. These statements can then be retrieved from this buffer.

Statement code

Statement codes are 1 character long statements which can be entered in the statement code column in F mode.

Statement code column

The statement code column is a field in the work window in which statement codes can be entered.

Statement line

A work window field in F mode. Entries in the statement line are interpreted as statements. The EDT statement symbol can normally be omitted.

Substitute character

When strings are converted from one character set to another, it is possible that characters in the source character set may not be present in the target
character set. In such cases, the substitute character is used (if it has been defined). If no substitute character has been defined then conversion is usually rejected.

Unicode mode

Unicode mode is an EDT operating mode. The extended EDT V17.0A functions are only available in Unicode mode, i.e. only in this Unicode mode is it possible to process Unicode files, can records exceed 256 bytes in length, are local character sets available etc.
However, this mode is not compatible with EDT V16.6B in all respects. In particular, there are incompatibilities in terms of the subroutine interface. In addition, a large number of inconsistencies have been eliminated.

Unicode substitute representation

In both statements in literals and in data, EDT permits the substitute representation of Unicode characters through the specification of the associated `UTF16` code. This makes it possible to enter all the (supported) characters via an escape character even if the character set for the statement or the data is not a Unicode character set.

User statement symbol

A user statement symbol is a special character which identifies user statements which are executed using external statement routines.

Wildcards

Wildcards are placeholders for groups of characters in a search string. Here, the `asterisk` (default value `*`) stands for a string of any length (including an empty string) and the `slash` (default value `/`) for precisely one character.

Work file

In EDT, data is always entered and processed in a work file. In work files, it is possible, for example, to insert, edit and delete data. The content of work files can be displayed on screen. If it is necessary to process the content of a file (DMS file, library element or POSIX file) then this content must first be transferred to a work file. After processing, the content of a work file can be written back to a file.
EDT is able to manage 23 work files. The work files are organized into records to which line numbers are assigned.

Work file, active

An **active** work file is a work file which contains a @DO procedure which is currently being executed. If there are nested @DO procedures then multiple work files may be active.

Work file, current

A single work file is the current work file. Data is entered in this work file and statements are effective within it. In F mode, a section of the current work file is displayed on the screen.

Work file, empty

An **empty** work file is a work file which contains no records. However, an empty work file may also contain properties which do not correspond to the initial state, for example it can be considered to be occupied or linked to a file. A work file is only reset to its original state following a @DELETE statement (format 2) or other statements which completely delete work files either implicitly or explicitly.

Work mode

EDT provides two work modes for data processing: line mode (L mode) and full screen mode (F mode).
In L mode, only one screen line is available for the entry of data and statements at any time.
In F mode, the entire screen is available for the entry of data and statements.The work modes should not be confused with the EDT operating modes (compatibility mode and Unicode mode). While the work modes relate to differences in the way data is displayed and processed, the operating modes represent different EDT environments with a restricted or extended function scope.

Work window

In F mode, the current work file is displayed on the screen. In this case, the screen is subdivided into fields with different functions. Alongside the data window in which the content of the current work file is displayed, the work window contains a statement line and a statement code column together with other elements.

10 Related publications

You will find the manuals on the internet at http://bs2manuals.ts.fujitsu.com . You can order printed copies of those manuals which are displayed with an order number.

[1] **EDT Unicode Mode**
Statements
User Guide

[2] **EDT**
Statements
User Guide

[3] **EDT**
Subroutine Interface
User Guide

[4] **SDF**
SDF Dialog Interface
User Guide

[5] **SDF-P**
Programming in the Command Language
User Guide

[6] **XHCS**
8-Bit Code and Unicode Processing in BS2000/OSD
User Guide

[7] **JV**
Job Variables
User Guide

[8] **BS2000/OSD-BC**
Commands
User Guide

[9] **LMS**
SDF Format
User Guide

[10] **POSIX**
Basics for Users and System Administrator**s**
User Guide

[11] **POSIX**
Commands
User Guide

[12] **ASSEMBH**
Reference Manual

[13] **ASSEMBH**
User Guide