English

FUJITSU

FUJITSU Software

BS2000 OSD/BC V10.0

Executive Macros

User Guide

Edition November 2015

Comments… Suggestions… Corrections…

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

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

Certified documentation according to DIN EN ISO 9001:2008

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 9001:2008.

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

Copyright and Trademarks

Contents

1 Preface

The present manual deals with all the macros which the user can issue to the BS2000 Executive and system services, and also describes the macros for terminal access.

1.1 Objectives and target groups of this manual

The manual addresses all BS2000 assembly language programmers.

The user should be familiar with the Assembler language and the use of macros. The “Assembler Instructions (BS2000)” manual [1] and “ASSEMBH” reference manual [2] can be used for this purpose. The present manual summarizes the most important information on the use of macros (chapter “Use of macros” on page 17).

The user should also have practical experience of BS2000. Appropriate information is provided in the “Introduction to System Administration” manual [10].

1.2 Summary of contents

The chapter “BS2000 Components” describes how the executive is embedded in the components of BS2000. It also explains the differences between job, task and process.

The basic syntax required to create a macro is described in the chapter “Use of macros”.

The chapter “Application areas and brief descriptions” lists the task areas for which macros are described in this manual.
The macros are divided into groups on the basis of their functions, which are explained in a brief description. Wherever required, the application of a functional group, i.e. the interaction of several macros, is described in greater detail.

The chapter “Description of the macros” describes all Executive macros and a selection of macros for other components in alphabetical order.
The DSECT of the macro is shown in some cases to aid understanding. Short program examples supplement the descriptions.

A macro described here can be looked up:

- alphabetically, via its name (running titles, table of contents, index and appendix, page 1142);
- via its function (table of contents and page 47);
- via its SVC number (appendix, page 1148).

The appendix describes macros supported only for reasons of compatibility. It then lists all the macros followed by a table of function key codes.

At the end of the manual you will find a number of chapters containing lists that will make it easier for you to work with the manual.

Readme file

The functional changes to the current product version and revisions to this manual are described in the product-specific Readme file.

Readme files are available to you online in addition to the product manuals under the various products at *http://manuals.ts.fujitsu.com*. You will also find the Readme files on the Softbook DVD.

Information under BS2000

When a Readme file exists for a product version, you will find the following file on the BS2000 system:

```
SYSRME.<product>.<version>.<lang>
```

This file contains brief information on the Readme file in English or German (<lang>=E/D). You can view this information on screen using the `/SHOW-FILE` command or an editor. The `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` command shows the user ID under which the product’s files are stored.

Additional product information

Current information, version and hardware dependencies, and instructions for installing and using a product version are contained in the associated Release Notice. These Release Notices are available online at *http://manuals.ts.fujitsu.com*.

1.3 Changes since the last edition of the manual

The amendments to BS2000 OSD/BC V10.0 have been introduced.

Macros for subsystems with separate version numbers are described for the following versions: BLSSERV V2.8A, TIAM V13.2A, VTSU V13.3A, see page 167.

Further important amendments

Macro	**Amendments**
ENTER	The description has been revised
NSIINF	New operand SRVUNIT, new operand value INFO=MIGCOUNT
STXIT	New operand MIGRATE; new STXIT event class “live migration”
VMGINF	New operand SRVUNIT

1.4 Notational conventions

The following abbreviated names are used in this manual:

- **BS2000 servers** for the servers with /390 architecture and the servers with x86 architecture.
  These servers are operated with the corresponding BS2000 operating system.
- **Servers with /390 architecture** for the Server Unit /390 of the Fujitsu Server BS2000 SE Series and the Business Servers of the S Series
- **Servers with x86 architecture** for the Server Unit x86 of the Fujitsu Server BS2000 SE Series and the Business Servers of the SQ Series (x86-64 architecture)
- **SE servers** for the Business Servers of the SE Series (Server Units /390 and x86)
- **S servers** for the Business Servers of the S Series (/390 architecture)
- **SQ servers** for the SBusiness Servers of the SQ Series (x86-64 architecture)

In the examples the strings `<date>`, `<time>` and `<ver>` specify the current outputs for date, time and version when the examples are otherwise independent of the date, time and version.

The following typographical elements are used in this manual:

MACRO	Names of macros are highlighted in bold in running text
`input`	Inputs in examples are shown in bold typewriter font
`Output`	DSECTS, compiler lists or outputs in examples are shown in typewriter font
i	For notes on particularly important information

References to other publications within the text are given in abbreviated form followed by numbers; the full titles are listed in the “References” section at the back of this manual.

2 BS2000 Components

The BS2000 operating system comprises two main groups, a central system and the user programs (e.g. language processors, file editors and utility routines).

The central system can be controlled by the user via the command language or the macros of the operating system BS2000.

The central system consists of the following components:

- Executive
- Data Communication Methods (DCM)
- Data Management System (DMS)
- System services

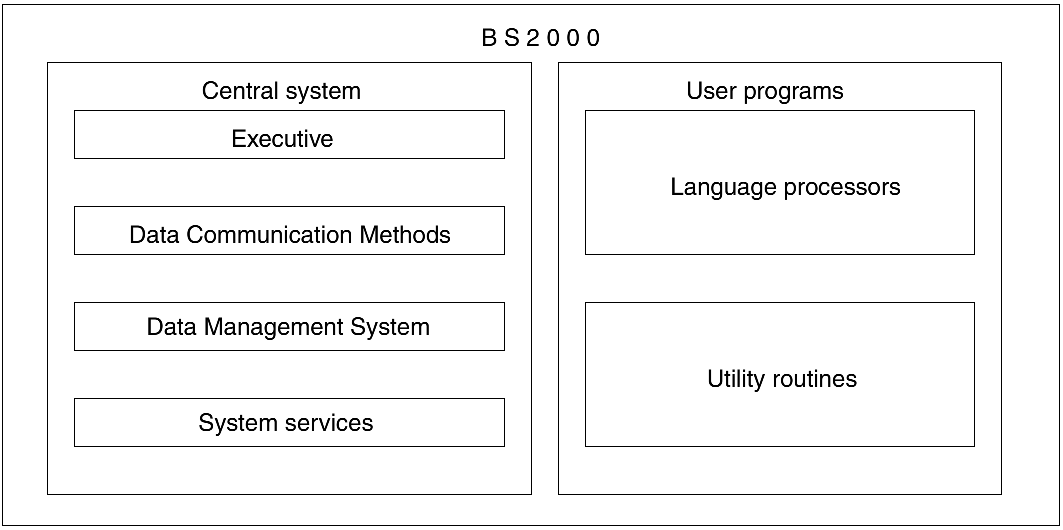


Figure 1: Components of BS2000

The **Executive** contains the central control routines of the operating system; it performs the following functions:

- controlling the execution of all jobs; e.g. all interactive, batch and SPOOL jobs
- management of virtual memory and real memory
- syntactical analysis of commands
- execution of spooling operations
- input from and output to operator consoles
- system resource accounting

BS2000 Executive macros enable program-specific use of the central control routines. This manual describes all the Executive macros.

The **Data Communication Methods** (DCM) perform the following functions:

- Data transfer between program and terminals or other programs.
- Management of the associated resources.

BS2000 macro calls to the communication access system control, among other things, operation of data display terminals. Of the macros for the communication access system, the use of the terminal access macros is explained in this manual. The “TIAM” manual [16] describes the functional scope of these macros. Other macros are described in the “DCAM” manual [15].

The **Data Management System** (DMS) includes routines supporting the following functions:

- File management (catalogs, stores, retrieves and deletes files
- Support of file access methods
- Input/output on peripheral devices (excluding consoles and terminals)

BS2000 Data Management System macros are provided for file, volume and device handling. Data Management System macros are described in the “DMS Macros” manual [7].

The **System services** include additional functions of the Control System, e.g.

- Advanced Interactive Debugger (AID)
- Dynamic Binder Loader (DBL)
- Linkage editor (BINDER)

The BS2000 Interactive Debugging Aid macros are used for error recovery in loaded programs, as this system function is capable of monitoring programs as well as affecting program execution.
In addition to the Executive macros, this manual describes all system service macros.

Jobs, tasks and processes

The concepts of job, task and process each have a specific meaning in BS2000.
Each of these terms is used to describe particular combinations of status and activity for a unit of work which has been submitted to the operating system.

Job: A sequence of commands, instructions and data which is contained between the SET-LOGON-PARAMETERS and EXIT-JOB commands. There is a difference between batch jobs and interactive jobs. In a batch job, the sequence of commands, instructions and data is read from a file; in an interactive job, this sequence is input interactively via the data display terminal. Job management assigns a job class to the job, and puts it into the appropriate job queue. When it enters the system, the job is given a job number (called the task sequence number, or TSN) by which it can be addressed during the time it remains in the system.

```
/SET-LOGON-PARAMETERS   ⎫
    :                   ⎪
Sequence of commands,   ⎪
instructions and data   ⎬  Job
    :                   ⎪
/EXIT-JOB               ⎭
```

Task: From the viewpoint of the operating system, a job becomes a task once system resources (CPU, memory, devices) are assigned to it. The task is controlled by task management, and a task control block (TCB) is created for it.

Process: The activities which are executed at program or module level within a task are referred to as the processes of the task. Each process has a process control block (PCB), which is used to record the exact state of a program if it is interrupted. The processes in a task are coordinated using the task control block.

The diagram below will clarify the relationship between the concepts of job, task and process:

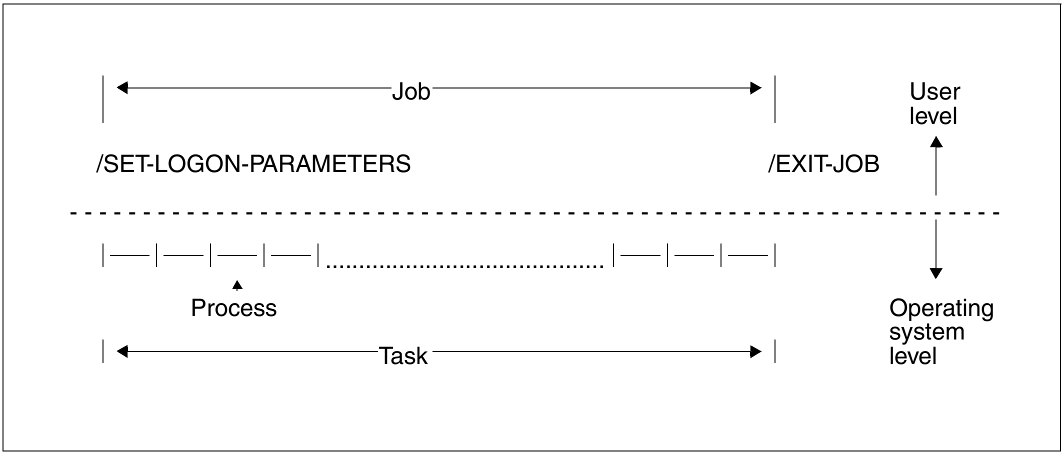


Figure 2: Jobs, tasks and processes

3 Use of macros

3.1 Macro processing by the assembler

A macro is a statement in a source program. It consists of a mnemonic operation code (macro name) specifying the particular function to be performed, which may be followed by values in the form of operands that supply information required for the execution of the macro, or give more details on the function to be performed (e.g. **EXIT** CONTINU=YES). The operands that may or must be specified are discussed in the “Description of the macros” on page 167.

Like every other source program statement, a macro is processed by the assembler at assembly time. Macro processing results in the source program macro being replaced by the macro expansion. A macro expansion consists of a sequence of instructions and assembly statements which together perform the function specified by the macro. When reference is made to statements generated by a macro, the macro expansion is meant.

The macro expansion is generated by the assembler via the macro definition. Every macro requires a macro definition. Macros can therefore be issued only if there is a corresponding macro definition. Macro definitions can be written by the user (see the “ASSEMBH” manual [2]) or are supplied to the user as part of the operating system (such as those definitions whose macros are dealt with in the present manual). All available macro definitions are included in the macro library (see the “Utility Routines” manual [27]). A macro definition is the original specification that the assembler uses to generate the macro expansion.

The macro definition is modified by the operand specifications in the macro call, and current values are loaded.

The source program, in which all macros are replaced by their macro expansions, is then assembled into machine language. The macro expansions are included in the assembly listing, unless they are suppressed by the PRINT NOGEN assembly statement (see the “ASSEMBH” manual [2]). The function requested by the macro is not performed until the program is executed.

A macro can be defined as the function which is implemented through the interaction of macro and macro definition.

3.2 Syntactical representation of macro calls

Format of a macro call

The macro format comprises two areas.
The upper area contains the optional name field and the macro name.
The lower area contains the possible operands.

[name] macro name
<operand$_1$> ,<operand$_2$>

An entry in the **name field** is permitted; the name specified is the symbolic address of the first statement in the macro expansion. The user may use this address, for example, as a branch address or as a halt point when using the Interactive Debugging Aid.
The call formats in the present manual generally omit the name field except in cases where a specification in a name field is of special significance. An example would be a macro of type S in L form (see page 30): The symbolic address specified in the name field is required for linking the data area with the instruction part of the macro (E form). Further examples are the **ARDS**, **CUPAB**, **DCSTA** and **TMODE** macros, where the default name of the generated dummy section may be replaced by the specification in the name field. Such macro formats also comprise the name field.

The **macro name** identifies the required macro. The dollar sign '$' is not used as the first character in user macros, because it is reserved for privileged macros.

The **operand field** may contain any number of operands, separated by commas, but it may also remain empty. The type and number of operands that may or must be specified is defined in the format description of each macro.

When calling a macro in an Assembler program, the name field, macro name and first operand must be separated by at least one space. Multiple operands must be separated by commas.

Format errors that are detected by the Assembler when the macros are being processed are included in the Assembler listing as MNOTE messages (see section “Macro language” in the “ASSEMBH” manual [2]).

Operand forms

Positional operands

Positional operands must be written in a specific order as they are interpreted by the Assembler on the basis of their position in the operand field.
Example: `MACRO A,B,C`

If the second operand (B) is deleted, the user must supply the second comma (immediately after the first comma) so as to maintain the proper position for the third operand (C):
`MACRO A,,C`

However, if the last positional operands are omitted, the delimiting commas need not be written. For example, if the operands B and C are omitted in the present example, the macro could be written as follows: `MACRO A`

Keyword operands

The keyword associated with a given keyword operand uniquely identifies that operand to the Assembler. Therefore, these operands can be written in any order. An operand value which originates from a defined set of allowed values is assigned to an operand by an equals sign.
Keyword operands have the following format: `<keyword>=<desired value>`
Example: `MACRO AREA=X,LENGTH=100`

Mixed operands

An operand field may contain a combination of positional and keyword operands; however, all positional operands must precede all keyword operands.
Example: `MAKRO A,B,C,AREA=X,LENGTH=100`

The rules for positional operand and keyword operand omissions also apply to mixed operand fields. Thus, if the operands B, C and AREA are omitted, the above example appears as: `MAKRO A,LENGTH=100`

Operand sublists

A sublist consists of one or more positional operands, each separated by commas.
The entire list must be enclosed in parentheses, and is considered to be one operand in that it occupies a single position in the operand field or is associated with a single keyword.
The contents of the sublist are processed similarly to positional operands.
Example: `(A,B,C)` or `(A)`
In the second example, the sublist consists of only one operand. In this case, the enclosing parentheses must still be written, even though there is only one element in the sublist, as otherwise it will not be recognized as a sublist.

Metasyntax

In the macro format, specific characters (metacharacters) and conventions are used; an overview is given in the following description.

Appears as	Meaning	Example
UPPERCASE LETTERS	Uppercase letters denote keywords or constants and must be entered by the user exactly as shown. Keywords must begin with * if both keywords and names or constants and variables can be specified as alternatives.	DIB FORCED=*YES

UPPERCASE LETTERS in boldface	Uppercase letters printed in boldface denote allowed abbreviations of keywords.	GLOBAL=**Y**ES
		The user must enter
		GLOBAL=YES oder
		GLOBAL=Y

lowercase letters	Lowercase letters denote data types of values, which can be specified by the user, or variables which, on entry, must be replaced by current values.	DIB=<var: pointer> FILE filename
< >	Angle brackets denote variables whose allowed values are described by the data types.	<var: pointer>

{ }	Braces enclose alternatives, i.e. one entry must be selected from the specifications enclosed.
	Exception: default values.
		TAPE=$\left\{\begin{array}{l}\text{YES}\\\text{NO}\end{array}\right\}$
		The user must enter
		TAPE=YES or
		TAPE=NO

/	The slash denotes a choice between alternatives; it has the same function as braces.	FORCED=*NO/*YES
		The user must enter
		FORCED=*NO or
		FORCED=*YES

underlining	Underlining denotes the default value of an operand, which is the value the system assumes if the user makes no entry (= system preset).
	If an operand has no default value, specification of an operand is mandatory.
		FORCED=*NO/*YES
		The user must enter
		FORCED=*NO or
		FORCED=*YES
		(no specification implies FORCED=*NO)

[ ]	Square brackets enclose options, i.e. the entries may be deleted. When a comma is enclosed between square brackets in optional entries, it need only be written if the option is used. When it is outside the brackets it must be specified even if the option is not used (round brackets must be entered).	filename[,ERASE] The user must enter, for instance, FILE,ERASE or FILE or XYZ,ERASE etc.

Table 1: Macro syntax

Appears as	Meaning	Example

`list-poss(n)`	A list can be formed from the operand values following list-poss. n specifies the maximum number of list elements. If the list contains more than one element it must be enclosed in round brackets.	`FLAG=list-poss(3):`
		`*SLI/*SKIP/*DC`
		The user must enter
		`FLAG=*SKIP`
		`FLAG=(*SLI,*DC)`

`...`	Ellipses denote repetition, i.e. the preceding syntactical unit may be specified one or more times in succession.	`(filename,...)`
		The user must enter
		`FILE` or
		`(FILE,XYZ)` or
		`(FILE1,FILE2,FILE3)`
		etc.

␣	This character denotes a blank (X'40')	`STD␣`
		The user must enter
		`'STD '`
		(without inverted commas)

`=`	The equals sign links the operand name to the operand values associated with it.	`DATA=<var:pointer>`

Table 1: Macro syntax

Data types of the operand values

Data type	Character set	Remarks
c-string	EBCDIC characters	must be enclosed in inverted commas
integer	[+-] 0..2147483647	is a decimal number
var:	precedes specification of a variable. The type of variable follows the colon (see table “Data types of variables”.)	<var: var-type>
reg:	Register 0..15	(<reg: var-type>)

Suffixes to data types

Suffix	Meaning
n..m	for the integer data type, n..m means an interval is specified; n: minimum value m: maximum value
	for the c-string data type, n..m means a length specified in bytes; n: minimum length m: maximum length with n < m
n	in the c-string data type, n means a length specified in bytes; n must be the exact number

The operand values can be entered directly as a character string or integer (see data types c-string and integer) or indirectly via a variable (see data type var:). The following table contains the data types that are possible for variables.

Data types of variables

Data type	Description	Definition in the program
char: n	The variable is a character string of n characters. If no length is specified, it is assumed that n = 1.	CLn
int: n	The variable is an integer that occupies n bytes. If no length is specified, it is assumed that n = 1. Condition: n ≤ 4	FLn
enum-of E: n	The variable is the enumeration of E, which occupies n bytes. If no length is specified, it assumed that n = 1 (n ≤ 4).	XLn
pointer	The variable is an address or an address value.	A

3.3 Use of registers

Wherever mentioned in the text of this manual, registers are referred to as R0, R1, ... This serves to distinguish references to registers from other numerical references; it is also a way of referring to registers frequently used in programs.

The Control System macros use general registers R0, R1 and R15; in addition, R13 and R14 are used by the **SAVE** and **RETRN** macros.

Registers R0 and R1 contain either operands or the address of operands in the macro.

Register R14 is sometimes used as the return register. It contains the address of the next instruction in the user program following the macro.

After the execution of a number of macros, register R1 contains the address of the data area rather than its original contents.

Register R15 contains the error flag (return code). If an error occurs during the execution of a macro, information on the execution is stored in the rightmost byte of register R15, before control is returned to the user program. Newer macros either do not use register R15 or only use it as an additional way of storing the return code. In these cases, a field for storing the return code is reserved in the standard header (see page 43).

3.4 Return information and error flags (return codes)

Return information

The transfer of information to the calling program is an integral part of the function of a number of macros. This information is sometimes stored in register R1 or transferred to a program area whose address is specified in the macro. The mode of transfer of such information is indicated in the description of the relevant macro.

Error flags (return codes)

After a macro has been executed, the calling program is informed of the successful or unsuccessful result of the macro. This is effected by transferring a return code. Depending on the relevant macro interface, the return code may be transferred either in register R15 or in the standard header. Some macros allow a combination of these two cases.

1. **Transferring the return code in the standard header**

(Part of standard header)

c	c	b	b	a	a	a	a

aaaa = main code
bb = subcode1
cc = subcode2

The **main code** is transferred in the two rightmost bytes. It identifies the result of the function execution. **Subcode 1** is used for error classification. **Subcode 2** subdivides the error into error classes or contains additional diagnostic information. All parts of the return code are specified in hexadecimal notation. For information on the structure and contents of the standard header, see page 43.

2. **Transferring the return code in register R15**

R15:

b	b					a	a

aa = primary return code
bb = secondary return code

The **primary return code** is transferred in the rightmost byte of register R15. It indicates whether or not the function was executed successfully.
If no error occurred during execution, the rightmost byte contains the code X'00'. If an error occurred during the execution of a macro, the Executive places an error flag in the form of a different hexadecimal code in this byte (the three leftmost bytes each contain X'00' provided that nothing else has been specified explicitly).
In some cases, the information provided by the primary return code is supplemented by a **secondary return code** in the leftmost byte of register R15. This code provides more detailed information on the cause of the error. The secondary return code, too, is specified in hexadecimal notation. It is the user's responsibility to analyze this code and to take proper action.
The return code values and their meaning are given under “Return information and error flags” in each macro description.

If the code values of the primary return code are given with an increment of X'04' and a guaranteed maximum value is defined, the return code can be processed using a branch table (consisting of 4-byte branch instructions).
The return codes of many macros do not have such a fixed structure. Such return codes must be processed with explicit queries (compare instructions).

An example follows for each of these two types of processing.

Return code of the **OPCOM** macro

X'aa'	**Meaning**
X'00'	ITC participation has been started.
X'04'	Error in operand specification. ITC participation has not been started.
X'08'	ITC name is already assigned. ITC participation is not started.
X'0C'	No system memory available for starting ITC or the system-internal size for receive queues is exceeded. ITC participation is not started.
X'10'	ITC participation has already been declared.

Example of return code processing with a branch table

```
  RCTAB    START
           PRINT NOGEN
           BALR  3,0
           USING *,3
  *        :
           OPCOM ITCNAME       * Declare ITC participation
1                 *,MACRO: OPCOM, VERSION: VER041
           B     RS00(15)
  CONTINUE EQU   *
  *        :
  END      TERM
  RS00     B     CONTINUE      * R15=00: No error handling
           B     OPERR         * R15=04
           B     NAMEERR       * R15=08
           B     MEMERR        * R15=0C
  EXIST    NOP   EXIST         * R15=10:
  *                              EXISTING ITC PARTICIPATION handling
           B     END
  OPERR    NOP   OPERR         *         OPERAND ERROR handling
           B     END
  NAMEERR  NOP   NAMEERR       *         DUPLICATE NAME handling
           B     END
  MEMERR   NOP   MEMERR        *         MEMORY ERROR handling
           B     END
           END
```

Example of return code processing with explicit query

```
  RCEXPL   START
           PRINT NOGEN
           BALR  3,0
           USING *,3
  *         :
           OPCOM ITCNAME       * Declare ITC participation
1                 *,MACRO: OPCOM, VERSION: VER041
           LTR   15,15
           BZ    CONTINUE      * R15 = X'00'
           C     15,=F'4'
           BE    OPERR         * R15 = X'04'
           C     15,=F'8'
           BE    NAMEERR       * R15 = X'08'
           C     15,=F'12'
           BE    MEMERR        * R15 = X'0C'
           C     15,=F'16'
           BE    EXIST         * R15 = X'10'
  *        :                   ***
  *        :                   * Handling of other return codes
  *        :                   ***
  CONTINUE EQU   *
  *        :
  END      TERM
  *
  OPERR    NOP   OPERR         * OPERAND ERROR handling
           B     END
  NAMEERR  NOP   NAMEERR       * DUPLICATE NAME handling
           B     END
  MEMERR   NOP   MEMERR        * MEMORY ERROR handling
           B     END
  EXIST    NOP   EXIST         * EXISTING ITC PARTICIPATION handling
           B     END
  *        :
           END
```

3.5 Macro expansion

Macros are implemented by supervisor calls (SVCs). In a few cases, the **CALL** macro for instance, the macro expansion contains no SVC. The supervisor call causes an SVC interrupt which activates the interrupt analysis. With the aid of an SVC index table the interrupt analysis determines the program module associated with the relevant macro. The validation of parameter values and the transfer to the calling program of information on any errors or on the processing status are controlled by the program module itself.

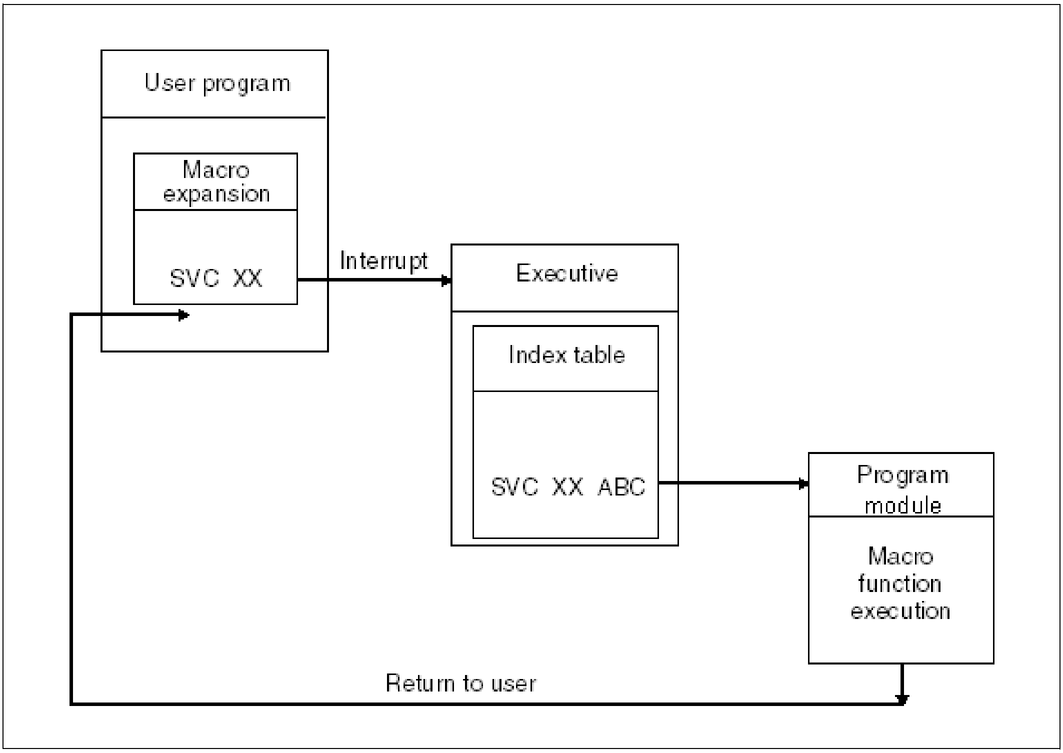


Figure 3: Flow diagram of SVC processing

3.6 Types of macro

To avoid any confusion it should be noted that the term “macro type” is not intended to differentiate the concepts of “action macro” and “definition macro”, which relate to the function of a macro:

An **action macro** is a macro that is expected to perform specific actions. The **AREC** macro, for example, is used to write a user acoounting record (see page 205).

A **definition macro** is a macro that is expected to supply definitions (addressing aids, DSECTS) rather than to perform actions. The **CUPAB** macro (page 377), for example, is used to generate symbolic names for the purpose of addressing operand tables.

The **type** classification of macros relates to the mode of **operand transfer**. There are R-type macros (operand transfer via registers), S-type macros (operand transfer via storage) and O-type macros (macros not classified under any type).
S-type macros can be either action macros (with operand MF=E) or definition macros (with operand MF=D).

3.6.1 O-type macros

There are a number of macros that cannot be classified as either type R or type S. They are simply referred to as “other macros” and are specified as O-type macros in the macro descriptions.

Examples of O-type macros are macros with an operand field allowing the specification of **one** register (frequently R1) that contains the start address of a parameter area.

The parameter area is defined in the data section of the program (DC statements) and contains the operand values.

3.6.2 R-type macros

MACRO
operand1 / (r1) ,operand2 / (r2)

A macro is of type R if all required operand values can be loaded into registers R0 and R1, which are provided for this purpose. An R-type macro does not therefore generate a parameter area.

Not all macros use both registers (e.g. the **DELFEI** and **RSOFEI** macros). The parameters may be specified directly as operands of the macro or may be contained in registers R0 and R1.

If `operand1` and `operand2` are specified, macro expansion causes the specified values to be loaded into registers R0 and R1.
If the registers are specified, the values of the operands `operand1` and `operand2` must have been loaded into registers R1 and R0 before the macro is called. This procedure is referred to as “register notation”.

Address operands in R-type macros can always be written as explicit or implicit addresses, i.e. in the form: base, index, displacement.
If the macro has only one optional operand, then any comment desired must be preceded by a “,” (comma).

Example: `CLCOM ,comment`

3.6.3 S-type macros

Operand values specified in S-type macros are transferred to the functional module in the form of a data area. The data area is part of the macro expansion. It contains the data definitions and storage definitions (DC and DS statements) required for transferring the operand values.

S-type macros support specification of the MF operand (see page 31). There are various ways of specifying this operand, depending on the functionality of the different macros.
There are three **MF formats**:

MF format	**for**	**Special features**
MF format 1	Macros with 24-bit interface: return code in register R15 Macros with 31-bit interface: return code in register R15 or (if available) in standard header	MF=S is the presetting (default format) for the MF operand.
MF format 2	Macros with 31-bit interface: return code in standard header	
MF format 3	Macros with 31-bit interface: return code in standard header	There is no particular presetting for the MF operand.

Table 2: MF formats for the S-type macros

Detailed diagrams of all three MF formats are given below, followed by a description of the operands and operand values.
Examples of macros in the S, D, E, M and L form are given on page 35.
The different forms are described on page 31.

MF format 1

[opaddr] MACRO

$$\left\{\begin{array}{l} \text{[MF=S] [,op}_1\text{,...,op}_n\text{]} \\ \text{MF=}\left\{\begin{array}{l}\text{L}\\ \text{(L,pre)}\end{array}\right\} \\ \text{MF=}\left\{\begin{array}{l}\text{D}\\ \text{(D,pre)}\end{array}\right\} \\ \text{MF=}\left\{\begin{array}{l}\text{(C,pre)}\\ \text{C}\end{array}\right\} \\ \text{MF=(E,}\left\{\begin{array}{l}\text{addr}\\ \text{(r)}\end{array}\right\}\text{)} \end{array}\right\}\text{[,PARMOD=24 / 31]}$$

The default format MF=S may not be specified explicitly for most macros in MF format 1. Any exceptions are indicated in the description of the macro (see also the operand description for MF=S, page 31).
With the 24-bit interface, prefix notation (e.g. (C,pre)) may not be used for the C/D/L form. Any exceptions are indicated in the description of the macro involved.

MF format 2

[opaddr] MACRO

$$\left\{\begin{array}{l} \text{[MF=}\underline{\text{S}}\text{] [,op}_1\text{,...,op}_n\text{]} \\ \text{MF=L [,op}_1\text{,...,op}_n\text{][,PREFIX=p]} \\ \text{MF=M,op}_1\text{,...,op}_n\text{[,PREFIX=p][,MACID=mac]} \\ \qquad\qquad\qquad\text{[,PARAM=addr / (r) / <var: pointer> / <reg: pointer>]} \\ \text{MF=D[,PREFIX=p]} \\ \text{MF=C[,PREFIX=p][,MACID=mac]} \\ \text{MF=E[,PARAM=addr / (r) / <var: pointer> / (<reg: pointer>)]} \end{array}\right\}$$

MF format 3

[opaddr] MACRO

MF=L [,op_1,...,op_n][,PREFIX=p]
MF=M,op_1,...,op_n[,PREFIX=p][,MACID=mac]
MF=R,op_1,...,op_n[,PREFIX=p][,MACID=mac]
MF=D[,PREFIX=p]
MF=C[,PREFIX=p][MACID=mac]
MF=E[,PARAM=addr / (r) / <var: pointer> / (<reg: pointer>)]

opaddr
Assembler name: If MF=L, identifies the address of the data area, otherwise optional. opadr can be used to address the data area with MF=(E,adr) or MF=M,PARAM=.

op_1,...,op_n
represent functional operands to be specified.

PARMOD=
controls macro expansion. Either the 24-bit or the 31-bit interface is generated.

24
The 24-bit interface is generated. Data areas and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data areas and instructions use 31-bit addresses (address space ≤ 2 Gb). Data areas start with the standard header.

MF=
determines the type of macro generation. Seven forms of macro can be distinguished, depending on the value specified for the MF operand:

S (default form: presetting for MF formats 1 and 2):
This operand value may not be specified in format 3.
MF=S may *not* be specified *explicitly* for most macros in MF format 1, i.e. the default form is selected by omitting the MF operand. MF=S is *not* included in the macro format for these macros. MF=S is included in the description of the call format for macros that allow MF=S to be specified explicitly.
First the instruction part is generated and then the data area, taking the operand values specified in the macro into account. The data area contains no field names and no explanatory equates. Initialization values are entered in the standard header.

C (C form)
Only the data area is generated. Each field is assigned a field name and explanatory equates, if required. The data area ends with a length equate. Initialization values must usually be entered in the standard header by the user.

(C,pre)
This specification is permitted only in format 1. The user can define the first characters of the field names and equates by specifying a prefix pre.
pre = 1..4 characters.

C [,PREFIX=p][,MACID=mac]
This specification is permitted only in MF formats 2 and 3. The PREFIX operand allows the user to define the first character of the field names and equates. p = 1 letter.
The MACID operand allows the user to define the second, third and fourth characters of the field name and equates. mac = 1..3 characters.
No other operands are evaluated in the C form.

D (D form)
A DSECT is generated. Each field is assigned a field name and explanatory equates, if required. The DSECT ends with a length equate. No switch to the initial location counter is effected.
The **DSECT** describes the structure of a memory area but does not occupy any memory space itself. The symbolic name specified with DSECT is entered in an ESD record (External Symbol Dictionary record). The location counter is set to zero.

(D,pre)
This specification is permitted only in MF format 1. The user can define the first characters of the field name and equates by specifying a prefix pre.
pre = 1..4 characters.

D [,PREFIX=p]
This specification is permitted only in MF formats 2 and 3. The user can define the first character of the field names and equates by specifying a prefix p. p = 1 letter.
No other operands are evaluated in the D form.

L (L form)
Only the data area is generated, taking the operand values specified in the macro into account. The data area contains no field names and no explanatory equates. The initialization values are entered in the standard header. The macro is contained in the definition section of the program. If it contains variable data, this form of macro must not be written in the reentrant portion of the program when shared-code programming is employed. The data area is initialized with constant values in the reentrant portion of the program, copied to a data area specific to the program run before the E-form call and modified there if necessary. Modification can be carried out, for example, with the M form, if it is available for the interface involved.

E (E form)
Only the instructions required for calling the functional module are generated. The instruction part usually ends with an SVC. The macro must indicate the address of the data area containing the operand values.

(E,addr) / (E,(r))
This specification is permitted only in MF format 1.
addr = Assembler name (address of the data area).
r = register containing the address of the data area. The register must be loaded with this address value before the macro is called.

E [,PARAM=adr / (r)]
E [,PARAM=<var: pointer> / (<reg: pointer>)]
The PARAM operand specifies the address of the data area. This specification is permitted only in MF formats 2 and 3.
addr = Assembler name (address of the data area).
r = register containing the address of the data area. The register must be loaded with this address value before the macro is called.
If no other specification is made, the default setting is: PARAM = (1)
No other operands are evaluated in the E form.

M (M form)
This operand value may be specified only in MF formats 2 and 3.
Instructions (e.g. MVCs) are generated which, while the program is running, use the operand values which are specified in the macro to overwrite fields in a data area already initialized with MF=L or, in the case of shared code programming, in a copy of the data area initialized with MF=L local to the program run. This is how the M form conveniently enables the operand values with which a macro is called to be **dynamically** matched with the program run.
If MF=M is specified, no default values are accepted for functional operands, i.e. all operands must be specified explicitly
Since the instructions generated with MF=M use the symbolic addresses and equates of the C form or D form, it must be established when using the M form that these names are available for addressing the data area to be modified. It is particularly important to ensure that for a macro with MF=M any PREFIX and MACID operands are specified with the same values as those in the associated MF=C or MF=D call.

M [,PREFIX=p][,MACID=mac]
The PREFIX operand allows the user to define the first character of the field names and equates.
p = 1 letter.
The MACID operand allows the user to define the second, third and fourth characters of the field name and equates.
mac = 1..3 characters.

M [,PARAM=addr / (r)]
M [,PARAM=<var: pointer> / (<reg: pointer>)]
The PARAM operand specifies the address of the data area. This specification is permitted only in MF format 2.
addr / <var: pointer> = Assembler name (address of the data area).
r / (<reg: pointer>) = register containing the address of the data area. The register must be loaded with this address value before the macro is called.
Default setting: PARAM = (1)

R (R form)
This operand value may be specified only in MF format 3.
The operand values (of output parameters) specified by means of functional operands are read from the data area and stored in variables of the application program.
Since the instructions generated for this purpose use the symbolic addresses and equates of the C form or D form, it must be established when using the R form that these names are available for addressing the data area to be modified. It is particularly important to ensure that for a macro with MF=R any PREFIX and MACID operands are specified with the same values as those in the associated MF=C or MF=D call.

R [,PREFIX=p][,MACID=mac]
The PREFIX operand allows the user to define the first character of the field names and equates.
p = 1 letter.
The MACID operand allows the user to define the second, third and fourth characters of the field names and equates.
mac = 1..3 characters.

Example 1: RDATA macro (MF format 1) with S form (standard form)

```
  RDATA1   START
           LDBASE R3,0
1                 *,MACRO: LDBASE, VERSION: VER021                    021
1          ##BALR R3,0                                                020
2          BASR  R3,0                                                 012
           USING *,R3
  RDATA1   AMODE ANY
  RDATA1   RMODE ANY
           GPARMOD 31
1                 *,MACRO: GPARMOD, VERSION: VER121
           PRINT GEN
           RDATA INAREA,STOP ——————————————————————————————————————————— (1)
1          ##SPASS  S0004S,S0004D                                    A312
2          CNOP  0,4
2          BAS   1,S0004S           ADDRESS AND SKIP PARAMS
1 S0004D     DS OF                                                   A340
1          FHDR  UNIT=36,FUNCT=18,VERS=2
2          DS    0A
2          DS    0XL8                GENERAL OPERAND LIST HEADER
2          DC    AL2(36)             FUNCTION UNIT NUMBER
2          DC    AL1(18)             FUNCTION NUMBER
2          DC    AL1(2)              FUNCTION INTERFACE VERSION NUMBER
2          DC    X'FFFFFFFF'          Returncode is virgin
1 *
1          DC    A(STOP)                  ERROR ADDRESS
1          DC    AL4(INAREA)              READ IN AREA ADDRESS
1          DS    AL1(0)                   PLACE FOR I.EDIT BYTE 1
1          DS    AL1(0)                   PLACE FOR I.EDIT BYTE 2
1          DC    AL1(0)                   SYSDTA ASSIGNMENT
1          DC    AL1(0)                   FLAG BYTE 1
1          DC    AL2(L'INAREA)           LENGTH OF READ
1          DC    AL1(0)                  FLAG TABLE BYTE
1          DC    AL1(0)                  ASSIGNMENT CHANGE INDICATOR
1          DC    H'0'                    KEY-POSITION
1          DC    H'0'                    KEY-LENGTH
1          DC    AL4(0)                  VTSUCB ADDRESS
1          DC    AL2(0)                  INPUT TIMER VALUE            009
1          DC    H'0'                    RES_FOR_TIAM                 007
1 *
1          @DCEI DCEDIT=,MODE=,IGETFC=,ICFD=,                            C
1                ITRSUP=,ILINEND=,IGETBS=,                               C
1                IMANUAL=,ILCASE=,IHDR=,                                 C
1                IGETIC=,RDA1=-20,RDA2=-19
2          ORG   *-20
2          DC    AL1(0)
2          ORG   *+20-1
```

```
2          ORG   *-19
2          DC    AL1(0)
2          ORG   *+19-1
2                *,@DCEI      999    921011   53531002
1 S0004S     DS 0Y                                                A340
1          SVC   39                     SYSFILE SVC
1 *
           PRINT NOGEN
  STOP     TERM

  *
  INAREA   DS    CL104
  R3       EQU   3
           END
```

(1) The standard form is the default setting for the **RDATA** macro and is selected by omitting the MF operand. All operands required must be specified. The instruction part and data area are generated.

Example 2: RDATA macro (MF format 1) with E and L form

```
  RDATA2   START
           LDBASE R3,0
1                 *,MACRO: LDBASE, VERSION: VER021                 021
1          ##BALR R3,0                                             020
2          BASR  R3,0                                              012
           USING *,R3
  RDATA2   AMODE ANY
  RDATA2   RMODE ANY
           GPARMOD 31
1                 *,MACRO: GPARMOD, VERSION: VER121
           PRINT GEN
           RDATA MF=(E,PARLIST) ---------------------------------------  (1)
1          LA    1,PARLIST              LOAD ADDR PARAM LIST INTO R1
1          SVC   39                     SYSFILE SVC
1 *
           PRINT NOGEN
  STOP     TERM
  *
  INAREA   DS    CL104
           PRINT GEN
  PARLIST  RDATA INAREA,STOP,MF=L -------------------------------------  (2)
1 S0007D     DS 0F                                                A340
1 PARLIST  FHDR  UNIT=36,FUNCT=18,VERS=2
2          DS    0A
2 PARLIST  DS    0XL8               GENERAL OPERAND LIST HEADER
```

```
2          DC    AL2(36)             FUNCTION UNIT NUMBER
2          DC    AL1(18)             FUNCTION NUMBER
2          DC    AL1(2)              FUNCTION INTERFACE VERSION NUMBER
2          DC    X'FFFFFFFF'          Returncode is virgin
1 *
1          DC    A(STOP)                  ERROR ADDRESS
1          DC    AL4(INAREA)              READ IN AREA ADDRESS
1          DS    AL1(0)                   PLACE FOR I.EDIT BYTE 1
1          DS    AL1(0)                   PLACE FOR I.EDIT BYTE 2
1          DC    AL1(0)                   SYSDTA ASSIGNMENT
1          DC    AL1(0)                   FLAG BYTE 1
1          DC    AL2(L'INAREA)           LENGTH OF READ
1          DC    AL1(0)                  FLAG TABLE BYTE
1          DC    AL1(0)                  ASSIGNMENT CHANGE INDICATOR
1          DC    H'0'                    KEY-POSITION
1          DC    H'0'                    KEY-LENGTH
1          DC    AL4(0)                  VTSUCB ADDRESS
1          DC    AL2(0)                  INPUT TIMER VALUE           009
1          DC    H'0'                    RES_FOR_TIAM                007
1 *
1          @DCEI DCEDIT=,MODE=,IGETFC=,ICFD=,                            C
1                ITRSUP=,ILINEND=,IGETBS=,                               C
1                IMANUAL=,ILCASE=,IHDR=,                                 C
1                IGETIC=,RDA1=-20,RDA2=-19
2          ORG   *-20
2          DC    AL1(0)
2          ORG   *+20-1
2          ORG   *-19
2          DC    AL1(0)
2          ORG   *+19-1

2                *,@DCEI      999    921011   53531002
1 *
           PRINT NOGEN
  R3       EQU   3
           END
```

(1) The E form of the macro generates the instruction part of the **RDATA** macro. The data area with the desired operands starts at the symbolic address PARLIST.

(2) All desired operands are specified in the L form of the macro. The data area is generated.

Example 3: GTIME macro (MF format 3) with D, E, M and L form

```
  GTIME    START
           PRINT NOGEN
           BALR  R3,0
           USING *,R3
  GTIME    AMODE ANY
  GTIME    RMODE ANY
           LA    R5,GLIST ———————————————————————————————————————————— (1)
           USING DGLIST,R5
           LA    R13,SAVE
  *
  E1       GTIME MF=E,PARAM=GLIST,LINKADR=*NONE ———————————————————————— (2)
           MVC   TEXT,='Date: ' ———————————————————————————————————————— (3)
           MVC   DATE,NTIGDTIC
           WROUT OUTPUT,STOP
2                *,@DCEO      999    921011   53531004
  CLEAR    MVC   DATE,=CL10' '
  *
  M        GTIME MF=M,PARAM=GLIST,DAY=YES ——————————————————————————————— (4)
  *
  E2       GTIME MF=E,PARAM=GLIST,LINKADR=*NONE ———————————————————————— (5)
           MVC   TEXT,='Day:  ' ———————————————————————————————————————— (6)
           MVC   DAY,NTIGDYID
           WROUT OUTPUT,STOP
2                *,@DCEO      999    921011   53531004
  STOP     TERM
  *
  OUTPUT   DC    Y(OUTPUTE-OUTPUT)
           DC    X'404001'
  TEXT     DS    CL6
  DATE     DS    CL10
           ORG   DATE
  DAY      DS    CL2
           ORG
  OUTPUTE  EQU   *
  SAVE     DS    18F
  GLIST    GTIME MF=L,DATE=YES ——————————————————————————————————————— (7)
           PRINT GEN
  DGLIST   GTIME MF=D ———————————————————————————————————————————————— (8)
1 DGLIST   MFTST MF=D,PREFIX=N,MACID=TIG,ALIGN=F,                    C
1                DMACID=TIG,SUPPORT=(E,D,C,M,L),DNAME=TIG_MDL
2 DGLIST   DSECT ,
2                *,##### PREFIX=N, MACID=TIG #####
1 *   subcodes
1 NTIGERROR_IN_CALL      EQU   1          Error in Call
1 NTIGRNAP               EQU   32         no Action possible
1 NTIGWARNING_SITUATION  EQU   512        Warning Situation (SPL)
```

```
1 NTIGRWCS               EQU   2           Warning Situation (ASS)
1 *
1 *   GTIME-Parameter-Area
1 NTIGFHDR FHDR  MF=(C,NTIG),EQUATES=NO                    Standardheader
2 NTIGFHDR DS    0A
2 NTIGFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER

2 *
2 NTIGIFID DS    0A              0   INTERFACE IDENTIFIER
2 NTIGFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 NTIGFCT  DS    AL1             2   FUNCTION NUMBER
2 NTIGFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 NTIGRET  DS    0A              4   GENERAL RETURN CODE
2 NTIGSRET DS    0AL2            4   SUB RETURN CODE
2 NTIGSR2  DS    AL1             4   SUB RETURN CODE 2
2 NTIGSR1  DS    AL1             5   SUB RETURN CODE 1
2 NTIGMRET DS    0AL2            6   MAIN RETURN CODE
2 NTIGMR2  DS    AL1             6   MAIN RETURN CODE 2
2 NTIGMR1  DS    AL1             7   MAIN RETURN CODE 1
2 NTIGFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1 *   main return codes
1 NTIGRNIN               EQU   1           GTIME function not
1 *                                        initialized
1 NTIGRNSI               EQU   2           no season information
1 NTIGRPRV               EQU   16          no previous change date known
1 NTIGRPST               EQU   17          no later change date in past
1 *                                        known
1 NTIGRNCD               EQU   18          no change date known
1 NTIGRXIE               EQU   8           internal error concerning
1 *                                        xcs_mode
1 *
1 NTIGIB1                DS    AL1         indicator byte 1
1 NTIGIMU                EQU   X'80'       MODE = UTC
1 NTIGIFB                EQU   X'40'       FORMAT = BIN
1 NTIGICS                EQU   X'20'        not used
1 NTIGIFT                EQU   X'10'       FORMAT = TODR
1 NTIGIDW                EQU   X'08'       date wanted
1 NTIGIWW                EQU   X'04'       day wanted
1 NTIGITW                EQU   X'02'       TOD wanted
1 NTIGIZW                EQU   X'01'       zone wanted
1 NTIGIB2                DS    AL1         indicator byte 2
1 NTIGIRM                EQU   X'80'       resolution = microsec.
```

```
1 NTIGICN                EQU   X'40'       next change date demanded
1 NTIGICP                EQU   X'20'       previous ch.date demanded
1 NTIGIRF                EQU   X'10'       time reference
1 NTIGIMX                EQU   X'08'       global XCS-time on
1 NTIGICA                EQU   X'04'       announcement of chdate
1 *                                        demanded
1 NTIGRESERVED_2BITS     EQU   X'03'       not yet used
1 NTIGIRES               DS    XL2          indicator byte 3 & 4
1 NTIGDATE_UNION         DS    0XL16       date_union
1 NTIGDATE_SPL           DS    XL16        for SPL
1         ORG   NTIGDATE_UNION
1 *
1 NTIGDTI                DS    0XL16       date_iso4
1 NTIGDATE_UN            DS    0XL10       date union

1 *
1 NTIGDATE_1             DS    0XL10       date struct
1 NTIGDTIY               DS    CL4         year
1 NTIGDTI1               DS    CL1         hyphen1
1 NTIGDTIM               DS    CL2         month
1 NTIGDTI2               DS    CL1         hyphen2
1 NTIGDTID               DS    CL2         day
1 *
1         ORG   NTIGDATE_UN
1 NTIGDTIC               DS    CL10        date_char
1         ORG   NTIGDATE_UN+10
1 NTIGDTIJ               DS    CL3         julian date
1 NTIGDTIB               DS    CL1         blank
1 NTIGDYID               DS    CL2         weekday in ISO4
1 *
1         ORG   NTIGDATE_UNION
1 *
1 NTIGDTB                DS    0XL16       date_bin
1 *
1 NTIGDATE_2             DS    0XL6        date
1 NTIGDTBY               DS    H           year
1 NTIGDTBM               DS    H           month
1 NTIGDTBD               DS    H           day
1 *
1 NTIGDTBJ               DS    H           Julian date
1 NTIGFILL_6             DS    XL6         fill for weekday
1 NTIGDYBD               DS    H           weekday bin.: MO=0, DI=1, ...
1 *                                        SO=6
1 *
.
.
.
```

Runtime log

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,gtime), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,gtime))
%  ASS6011 ASSEMBLY TIME: 538 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 135 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=gtime
%  BLS0523 ELEMENT 'GTIME', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'GTIME', VERSION ' ' OF '<date> <time>' LOADED
Date: 2012-01-20
Day:  FR
```

(1) Register R5 is loaded with the address of the data area created with MF=L and used to address the parameter list.

(2) The call **GTIME MF=E** will generate the command section.
The data area with the operand values starts as of the symbolic address GLIST: The current date (DATE=YES) should be determined (see also point (7)).

(3) The output area will be filled in with the text “Date:”, and the contents of the field NTIGDTIC. The field NTIGDTIC is part of the DSECT and of the data area GLIST and contains the current date. The desired information will be output with the **WROUT** macro.

(4) The call **GTIME MF=M** will dynamically modify the GLIST data area. Additionally the current day (DAY=YES) should now be output.

(5) The call **GTIME MF=E** will generate the command section.
The data area with the operand values again starts as of the symbolic address GLIST: The current day (DAY=YES) should now be determined (see also point (7)).

(6) The output area will be filled with the text “Day:” and the contents of the field NTIGDYID. The field NTIGDYID is part of the DSECT and of the modified data area GLIST and contains the current day after a claa with MF=E has been issued. The desired information will be output with the **WROUT** macro.

(7) The call **GTIME MF=L** generates the data area for operand values. The data area starts as of the symbolic address GLIST. The first time **GTIME MF=E** is called, the information DATE will be queried (see point (2)). The second time **GTIME MF=E** is called, the data area has been changed with a previous call **GTIME MF=M**, so that the desired information is now also DAY (see points (4) and (5)).

(8) The DSECT for GTIME is generated.
The data area can be completed by using the field name of the DSECT. The field names begin by default with the characters NTIG.
The calculation of addresses in DGLIST once again begins with X'000000'. The subsequent offsets are addressed with the base register R5 (e.g. in MVCs).

3.7 Standard header

All macros which are new and, as a rule, all existing macros which which have been extended by the 31-bit interface make use of the standard header in order to identify their interface.

The standard header is an 8-byte field at the beginning of the data area containing the (standardized) interface name plus 4 bytes reserved for a return code. The standard header is generated and initialized, i.e. supplied with the valid values for UNIT, FUNCTION and VERSION, by the corresponding macro. When using an E-form macro with reference to the data area, the caller may have to initialize the standard header. Where required, details are given in the macro description.

Structure of the standard header:

Byte	Field contents and meaning
0 - 1	Name of the unit containing the requested function
2	Name of the function within the unit
3	Name of the version of the function
4	SUBCODE2 of the return code
5	SUBCODE1 of the return code
6 - 7	MAINCODE of the return code

Table 3: Standard header

The following list gives the standard return code values which apply to all macros:

SUB-CODE2	SUB-CODE1	MAIN-CODE	Meaning
X'00'	X'00'	X'0000'	The function has been performed without errors. There is no information in addition to the MAINCODE.
X'01'	X'00'	X'0000'	The function has been performed without errors. No further action was required.
X'00'	X'01'	X'FFFF'	The requested function is not supported (incorrect entry for UNIT or FUNCTION in the standard header). Unrecoverable error.
X'00'	X'02'	X'FFFF'	The requested function is not available. Unrecoverable error.
X'00'	X'03'	X'FFFF'	The specified version of the interface is not supported (incorrect entry for VERSION in the standard header). Unrecoverable error.
X'00'	X'04'	X'FFFF'	Data area is not aligned on a word boundary.
X'00'	X'41'	X'FFFF'	The subsystem is not present and must be explicitly generated.
X'00'	X'42'	X'FFFF'	The calling task is not connected to this interface; the connection must be explicitly established.

Table 4: Standard return codes

SUB- CODE2	SUB- CODE1	MAIN- CODE	Meaning
X'00'	X'81'	X'FFFF'	The subsystem is currently not available.
X'00'	X'82'	X'FFFF'	The subsystem is in the DELETE or HOLD state.

Table 4: Standard return codes

MAINCODE indicates the result of function execution. SUBCODE1 qualifies MAINCODE. SUBCODE2 further qualifies the error by means of error classes or contains additional diagnostic information.
With all new macros, the return code should be supplied exclusively in the standard header. Some macro interfaces allow the return code to be passed in register R15, either as an alternative or in addition to being supplied in the standard header. The return code field should be initially set to X'FFFFFFFF' in order to permit checking whether a return code has been transferred to the standard header or not.

Example: generation of standard header

```
           WROUT    START
           PRINT NOGEN
           BALR  3,0
           USING *,3
  WROUT    AMODE ANY
  WROUT    RMODE ANY
           GPARMOD 31
1                 *,MACRO: GPARMOD, VERSION: VER121
           PRINT GEN
           WROUT OUTPUT,STOP
1          ##SPASS S0002S,S0002D                                    A312
2          CNOP  0,4
2          BAS   1,S0002S           ADDRESS AND SKIP PARAMS
1 S0002D     DS OF                                                  A340
1          FHDR  UNIT=36,FUNCT=17,VERS=2
2          DS    0A
2          DS    0XL8                GENERAL OPERAND LIST HEADER
2          DC    AL2(36)             FUNCTION UNIT NUMBER
2          DC    AL1(17)             FUNCTION NUMBER
2          DC    AL1(2)              FUNCTION INTERFACE VERSION NUMBER
2          DC    X'FFFFFFFF'          Returncode is virgin
1 *
1          DC    AL4(STOP)                ERROR ADDRESS
1          DC    AL4(OUTPUT)              MESSAGE AREA ADDRESS
  *        :
```

3.8 Macro Command Language Processor macros

The Macro Command Language Processor (MCLP) allows (system) commands to be entered without exiting the program mode. The macro **CMD** is used to invoke the MCLP (with SVC 58_{16}) and to communicate the command name and operands. The MCLP carries out a syntax check before branching to the actual command processing routine itself. After the command has been executed, the program is resumed.

Some of the commands which can be invoked will terminate the calling program (see table 13 on page 317). The calling program is also terminated if (user-specific) commands defined using SDF-A and implemented by command procedures are called.

In addition to SDF commands, ISP commands may also be invoked. Input errors in SDF command names may be corrected in interactive mode.

Some of the BS2000 commands contained in the system syntax file of BS2000 OSD/BC cannot be called via the **CMD** macro, see table 12 on page 315.

Some of the commands have own macros. The following table compares these macros with the corresponding commands (macros described in the appendix are not included):

Macro	**Command**	**Function**
CDUMP2	CREATE-DUMP	Generate dump
CHKPRV	SHOW-PRIVILEGE	Query own job privileges
ENTER	ENTER-JOB	Submit a job
LGOFF	EXIT-JOB	Terminate job
MSGSHOW	SHOW-MSG-FILE-ASSIGNMENT	Output information on system or task-specific message files
MSGSINIT	MODIFY-MSG-FILE-ASSIGNMENT	Lock message file or add message file to message system
MSGSMOD	MODIFY-MSG-FILE-ASSIGNMENT	Lock message file or add message file to message system
NKDINF	SHOW-DEVICE-DEPOT SHOW-DEVICE-CONFIGURATION SHOW-DEVICE-STATUS SHOW-DISK-DEFAULTS SHOW-DISK-STATUS SHOW-MOUNT-PARAMETERS SHOW-RESOURCE-ALLOCATION SHOW-RESOURCE-REQUEST SHOW-TAPE-STATUS	Output information on the allocation and availability status of configuration and mounted volumes
NSIINF	SHOW-SYSTEM-INFORMATION	Output system information
NSIOPT	SHOW-SYSTEM-PARAMETERS	Output system parameters

Table 5: Commands with their own macros

Macro	**Command**	**Function**
RDUID	SHOW-JOB-STATUS	Read user ID
SINF	SHOW-SYSTEM-INFORMATION SHOW-SYSTEM-PARAMETERS	Output system information and system parameters
SRMUINF	SHOW-USER-ATTRIBUTES	Output information from user catalog
STAMCE	SHOW-MASTER-CATALOG-ENTRY SHOW-PUBSET-PARAMETERS	Output MRSCAT entries Output pubset information
SWITCH	MODIFY-JOB-SWITCHES MODIFY-USER-SWITCHES SHOW-JOB-SWITCHES SHOW-USER-SWITCHES	Set job switches Set user switches Query job switches Query user switches
SYSFL	ASSIGN-SYSDTA ASSIGN-SYSLST ASSIGN-SYSOUT REMOVE-TASKLIB SET-TASKLIB	Assign system files
SYSTA	SHOW-SYSTEM-FILE-ASSIGNMENTS	Output system file assignments
TCHNG	MODIFY-TERMINAL-OPTIONS	Modify terminal attributes

Table 5: Commands with their own macros

4 Application areas and brief descriptions

4.1 Linking and loading

Macro	Brief description
ASHARE	Links and loads shared code, which may consist of a set of modules, into a memory pool
BIND	Calls the dynamic binder loader DBL to link and load one or more modules and continues the task optionally with the calling program or with the loaded module
CALL	Loads a segment that is not yet in memory and continues the task with the loaded segment. Subsequent segments within the same path of the overlay structure are loaded automatically
DSHARE	Unloads shared code from a memory pool
ETABIT	Generates or changes an entry for a symbol table transferred to the DBL in the ETABLE macro
ETABLE	Transfers a symbol table to the DBL; the table is integrated into the symbol table of the specified context
GETPRGV	Displays the program version previously selected by the user with the SELPRGV macro or the SELECT-PROGRAM-VERSION command
ILEMGT	Manages a list of ILEs (Indirect Linkage Entries)
ILEMIT	Generates a list entry for an ILE list which is used in the ILEMGT macro
LDSLICE	Loads the specified slice, that has been defined by the user in a link and load module (LLM), into main memory
LPOV	Loads the specified segment, even if it is already in memory, and continues the task optionally with the calling program or with a freely selectable module
PINF	Provides information on programs loaded with the LOAD-PROGRAM or START-PROGRAM command
SELPRGV	Determines which program version the DBL is to use if several versions of a program are loaded
SEGLD	Loads a segment, even if is already in memory, and continues the task optionally with the calling program or with any module. Subsequent segments within the same path of the overlay structure are loaded automatically

Macro	Brief description
UNBIND	During program execution, releases memory space occupied by an object that is no longer required and optionally unlinks CSECTs and ENTRYs within the object (i.e. external references to these symbols are then handled by the DBL as unresolved external references). The object can be a load unit, a link and load module (LLM) or an object module (OM)
VSVI1	Provides the user with information about entries in the tables of the DBL, in particular about the names of the contexts and also the names, load addresses, lengths and attributes of CSECTs, ENTRYs and COMMONs

The **TABLE** macro is still supported for compatibility reasons only. It is described on page 1137.

Detailed information on the link loader starter DBL and the BINDER functions can be found in the manuals “BLSSERV” [4] and “BINDER” [5].

4.2 Virtual address space

This section describes the structure of the virtual address space in BS2000 and how to convert virtual addresses into real addresses (address conversion). It also contains brief descriptions of the macros and detailed descriptions of the application areas “Working with virtual address space”, “Memory pools” and “Extended addressing with data spaces”.

4.2.1 Structure of virtual address space

Virtual address space is a series of virtual addresses in continuous ascending order, starting at 0.

The size of the virtual address space can be selected for servers with /390 architecture and may be up to 2 Gb, see figure 4 on page 51. The user address space for servers with x86 architecture is generated with 2 Gb and cannot be modified.

Since commands cannot be executed unless they and their operands are located in main memory, it is necessary to convert the virtual addresses of the address space into real main memory addresses (see page 53). This conversion takes place when the program is executed.

BS2000 divides virtual address space into pages for management purposes. A page consists of 4 Kb (4096 bytes), i.e. users can request memory for their programs in portions of 4 Kb.
Each task is assigned its own virtual address space when it is created. Virtual address space is divided into six memory classes with different attributes. Each page can be assigned to exactly one of these classes.
Classes 5 and 6 together represent the task-local part of the virtual address space, although only class 6 memory is addressable by users for their own programs and data (user address space). In class 5 memory, the system sets up tables that it requires for communication with the user task. Nonprivileged users cannot normally access this memory area. There are exceptions to this rule, e.g. nonprivileged DSSM subsystems that can also be loaded into class 5 memory. Classes 1 through 4 are privileged, although class 4 can contain subsystems and users' reentrant programs in addition to system tables. Classes 1, 2 and 3 are available only to the system.

Division into memory classes

Memory class	Attributes	Contents and availability
6	– nonresident (presetting) – assigned dynamically – task-local	– Programs and work areas of the (nonprivileged) user – Nonprivileged users have read and write access to this class
5	– nonresident – assigned dynamically – task-local	– System tables for linking the task to the system and to nonprivileged subsystems – Nonprivileged users do not normally have access to this class
4	– nonresident – assigned dynamically – system-global	– Pageable tables, dynamically loadable parts of the system and reentrant programs (shared code) – Nonprivileged users have read access to shared code
3	– resident – assigned dynamically – system-global	– Resident tables, dynamically loadable parts of the system and system work areas – May not be accessed by nonprivileged users
2	– nonresident – static – system-global	– Pageable tables and system modules – May not be accessed by nonprivileged users
1	– resident – static – system-global	– Resident tables and system modules – May not be accessed by nonprivileged users

Table 6: Definition of the memory classes

Virtual address space on servers with /390 architecture

The information in figure 4 shows the structure of the 2 Gb virtual address space in servers with /390 architecture (standard setting).

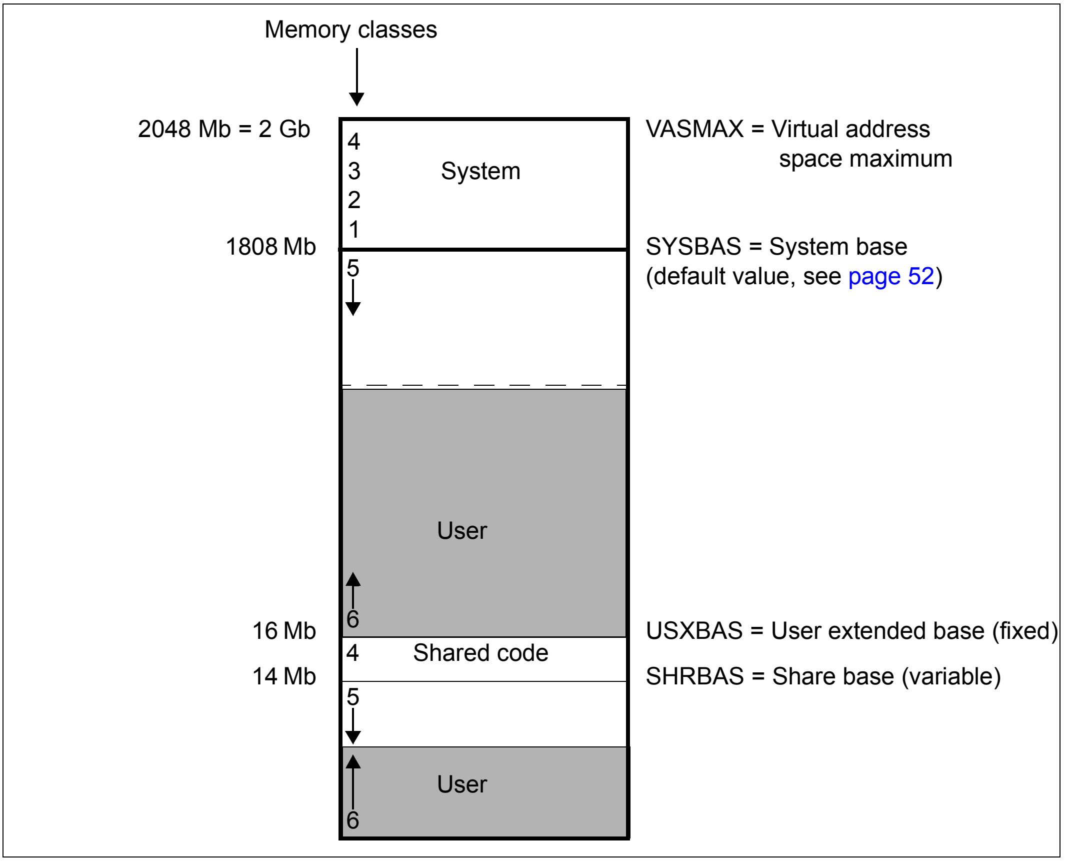

Memory classes
2048 Mb = 2 Gb
4
3
System
2
1
VASMAX = Virtual address space maximum
1808 Mb
SYSBAS = System base (default value, see page 52)
5
User
6
16 Mb
USXBAS = User extended base (fixed)
4
Shared code
14 Mb
SHRBAS = Share base (variable)
5
User
6

Figure 4: Structure of virtual address space (servers with /390 architecture)

Size of user address space

User address space (class 6 memory) and task-local system address space (class 5 memory) each have an area of address space above and below the 16-Mb boundary. There are no fixed boundaries between class 5 and class 6 memory. The boundaries can vary in either direction, depending on the memory requirements for each of the two memory classes. The maximum size of the user address space is also variable, therefore, and depends on the amount of class 5 memory required for the activities performed by the user task. However, 1/8 of the common area is always reserved for class 5 memory; class 6 memory can never occupy the entire area.

There are three other factors that determine the size of the user address space:

1. Setting the size of the user address space (servers with /390 architecture only)
   The user address space is generated with 1808 Mbytes.
   The procedure SYSPRC.BS2000-EXEC.version can be used to generate a BS2000 EXEC with a different user and overall address space using the BS2000 standard EXEC supplied as a basis, see the “System Installation” manual [11].
   Of the standard size of 1808 Mb (or 896 Mb or 448 Mb) for the task-local address space (which corresponds to a total address space of 2048 Mb (or 1024 Mb or 512 Mb)), 1/8 is reserved for class 5 memory. This leaves a size of 1568 Mb (or 770 Mb or 378 Mb) for class 6 memory above 16 Mb.

2. System initialization
   The size of the shared code area below 16 Mb can be set via the parameter service at system initialization time. The default setting is 2 Mb. In addition, during system initialization DSSM determines the total requirement for class 5 memory across all subsystems using SCOPE=*GLOBAL. A correspondingly large area is reserved for class 5 memory (in addition to the general share of 1/8), but this can be released again on a task-local basis using /RELEASE-SUBSYSTEM-SPACE. A total of approximately 12 Mb remains for the class 6 memory when SHRSIZE=2 Mb.

3. User catalog
   The user entry defines the contingent (in Mb) which is available to the user for allocations in the virtual address space. It covers the memory requests in class-6 memory of the user address space and in the data spaces which are created by the user.
   The maximum size of class 6 memory available to each user is entered in the user catalog. However, this value determines only the amount of class 6 memory required, not its location (above or below 16 Mb).
   In the output of the SHOW-USER-ATTRIBUTES command, the ADDRESS-SPACE-LIMIT field contains the maximum permitted size of class 6 memory for a user ID (see the “Commands” manual [19]).

4.2.2 Address conversion

Each virtual address space forms a domain. If an address space is not part of the system address space and has not been defined explicitly as shareable, the pages of the address space are accessible only within this address space. In other words, the pages are protected against access from other address spaces. This protection is achieved by means of address conversion, which always refers to the activated address space when converting a virtual address into a real address. On task initialization, a special hardware register (CR1) is activated. This register contains the base address and the length of the conversion table valid for this address space. (When a task is deinitialized, its address space is deactivated; switching a task therefore also switches the address space associated with it.)

A virtual address is composed of the segment number, page number (index) and byte number. BS2000 converts this address into a real address in two stages. Firstly, a page table is selected via the corresponding segment table (conversion table). This page table then contains the (real) main memory page number associated with the virtual page number. Address conversion takes place only for the page section (segment and page number) of an address; the real and virtual displacement within a page remains unchanged.

During address conversion, the varying size of hardware pages for different hardware architectures should be taken into account:
The size of virtual and real pages is 4 Kbytes for all BS2000 servers.

Accordingly, the structure of a virtual 31-bit address during address conversion will vary:
For all BS2000 servers, the page number is 8 bits and the offset 12 bits wide;
The segment number is 11 bits wide on all BS2000 servers.

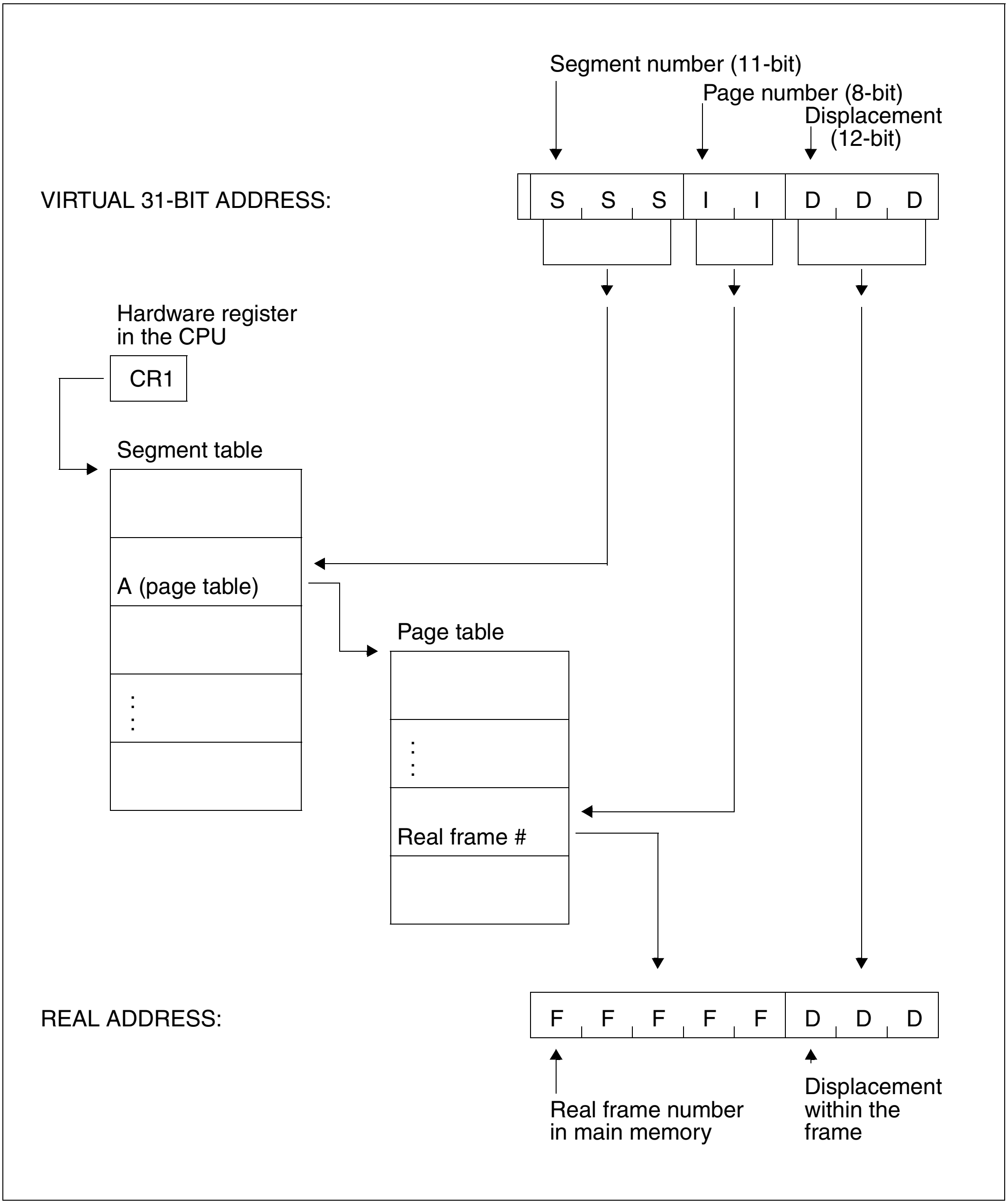


Figure 5: Address conversion for a 31-bit address

4.2.3 Working with virtual memory

Macro	Brief description
CSTAT	Changes the status of memory pages owned by a program (paging and read/write access)
MINF	Provides information about the allocation and size of class 6 memory or of a memory pool
RELM	Releases a contiguous memory area from the calling program
REQM	Requests (additional) memory space for the program

Pages in a memory pool (see page 56) can be requested using the **REQMP** macro and released using the **RELMP** macro.

4.2.4 Common memory areas shared by several users (Memory pools)

Macro	Brief description
CSTAT	Changes the status of memory pages of a program (paging and read/write access)
CSTMP	Changes the read/write access to a memory pool
DISMP	Terminates participation in a memory pool. The specified memory pool is deallocated if the calling program is the last (or only) participant
ENAMP	Sets up a memory pool, or enables participation in an existing memory pool
MINF	Provides information about allocation of pages and the size of a memory pool
RELMP	Releases (contiguous) memory space within a memory pool
REQMP	Requests (contiguous) memory space within a memory pool

Memory pool characteristics

A memory pool is an area in memory (class 6 memory) that can be shared by several users. Each of the participating users may read from or write to any page of the pool. A user who is not a participant has no access to the pool pages.

If a pool participant changes the contents or the status of the pool pages, all pool participants are affected. If a participant writes to a page of the pool, this data is available to all other participants. If a participant protects a pool page against overwriting (**CSTAT** macro), no pool participant can write to this page any more. Subroutines residing in the pool should be reentrant.

In order to coordinate accesses to the memory pool and to ensure efficient interworking, execution of the participating tasks should be synchronized. One way in which the user may accomplish this is by the (task) serialization method (see page 91).

Information on the memory pools currently created in the system and the joined tasks is provided by the BS2000 command `/SHOW-MEMORY-POOL-STATUS`, see the “Commands” manual [19].

Opening a memory pool

By means of the **ENAMP** macro, a user can set up a new memory pool or join an existing one (dependent on the MODE operand of the macro).

When creating a new pool, the user defines its name, scope and size. These definitions are binding for any further participants. The scope determines whether (and if so, which) other tasks may participate in the pool.

The size of the pool is limited by the amount of user memory available to the participants. The memory pool is created in units of 64K or 1 Mb. Memory pools with 64K units are always created below the 16-Mb boundary and will not be supported on a long-term basis. It is advisable to create memory pools with 1-Mb units only (enhanced performance). The **ENAMP** macro causes the system to reserve for the pool the appropriate number of entries in the segment table (see figure 6 on page 59).

A user joining an existing memory pool must accept the pool name, scope and size. The system provides an ID that can be used instead of the name to speed up processing. When opening a pool, each participant can specify an address at which the system can enter the ID.
When opening a pool, users may individually specify which part of their available address space is to be allocated to the memory pool by specifying a start address for the pool within this address space. Each participant may address the pool using a selected area of the address space. The same address need not be selected by all participants. The system supplies the participants with the virtual address of the first pool byte via register R1.

Closing a memory pool

The **DISMP** macro terminates the user's participation in the specified memory pool, thereby releasing that part of that user's address space that was allocated to the pool. If **DISMP** is called by the last remaining or the sole pool participant, the memory pool is deleted altogether. (The participant deleting the memory pool need not be identical with the participant who created the pool.)

Requesting memory in a memory pool

Using the **REQMP** macro, any participant in a memory pool can request memory in the pool in 4K pages. The system allocates the requested pages in the virtual memory area reserved for the pool (by **ENAMP**). Any pool participant can now write to and read from the allocated pages. The participant who submits the request (binding for all participants) specifies which, and how many, pages are to be allocated within the reserved pool area. The **MINF** macro provides information on the page allocation and size of the memory pool.

Releasing memory in a memory pool

Using the **RELMP** macro, any participant in a memory pool can release memory pages previously allocated to the memory pool. It is irrelevant which participant originally reserved the memory area and which portions of it were allocated. The participant that calls **RELMP** specifies how many and which of the allocated pages are to be released. The release is valid for all pool participants.

Page status in a memory pool

The **ENAMP** macro may be used to specify whether the memory area is to be resident. This may only be done by an authorized caller: the START-PROGRAM command for the program calling **ENAMP** must define the number of resident pages for the task. This value must be taken into account if the user defines the size of the memory pool in the **ENAMP** call and also specifies that the pool is to be resident.

Using the **CSTAT** macro, a pool participant can change the page status within user memory. The page status of a memory pool can be changed by means of **CSTAT** from pageable to resident only. A change in page status from resident to pageable is ignored for memory pools, which have been specified as resident by **ENAMP**.

The **CSTAT** macro is not restricted by memory pool boundaries. The range of pages influenced by **CSTAT** may begin outside of a pool and end within the pool, and vice versa. When the page status is changed from resident to pageable, only pages outside the resident memory pool are changed.

If the page status is changed from pageable to resident, all pool participants can access the resident pages. This change of status can only be performed by an authorized pool participant. (The START-PROGRAM command for the program calling **CSTAT** specifies whether and how many memory pages may be resident).
The user can also use **CSTAT** to set the access mode (read/write access) for the specified memory pages. The memory pages can be inside or outside a memory pool.

The (authorized) user can set the access mode for a memory pool with the **CSTMP** macro. This specification always applies to all pages of the memory pool. The following points should be noted:

- **CSTAT** has a lower priority than **CSTMP** for changing the access mode (read/write access).
  Write protection imposed with **CSTMP** cannot be removed with **CSTAT**.
- **CSTAT** is rejected if write protection was imposed with **CSTMP**.
- Write protection imposed with **CSTAT** can be extended to all pool pages with **CSTMP**.

Restrictions on the use of memory pools

- Checkpoint processing (**WRCPT** macro / RESTART-PROGRAM command) is illegal if a memory pool is open.
- The HOLD-TASK command (see “Commands” manual [19]) is rejected.
- If an unrecoverable main memory error occurs in a pool page, any program which attempts to access the errored pool page will be terminated.

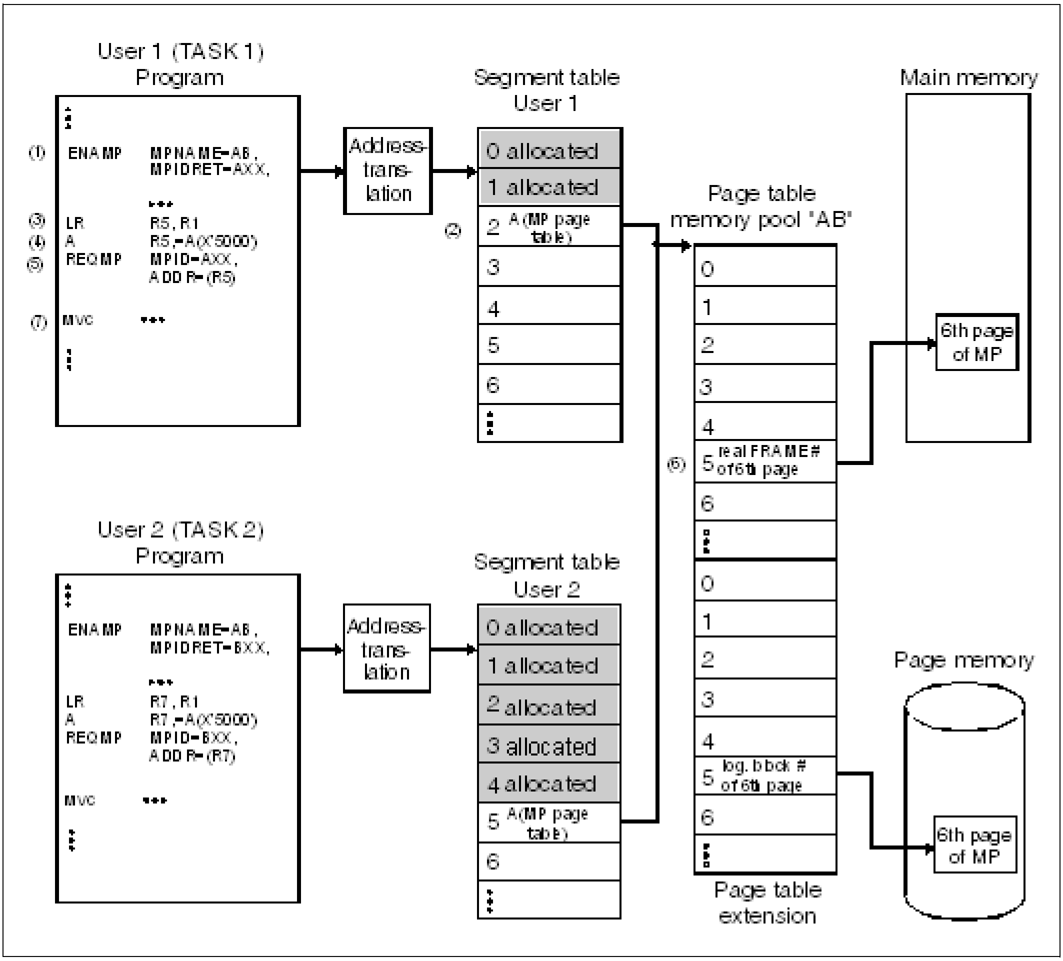


Figure 6: Establishing a connection to and accessing a memory pool page (servers with /390 architecture)

(1) The **ENAMP** macro establishes a connection with the memory pool AB.

(2) The virtual start address of the memory pool page table is entered in the next free entry of the segment table. In this case, the next free entry is the entry for the third segment. For user 1, therefore, memory pool AB starts at the virtual address X'00200000'. This virtual start address is stored in register R1.

(3) The contents of register R1 are loaded into register R5. Both registers now contain the virtual start address of memory pool AB for user 1.

(4) 6 pages (X'5000') are added to the virtual start address of memory pool AB and the results are stored in register R5.

(5) The **REQMP** macro then requests the 6th page of memory pool AB.

(6) The 5th entry in the memory pool page table contains the real frame number of the 6th page of memory pool AB in main memory. The 5th entry in the page table extension contains the logical block number of the 6th page in page memory.

(7) The 6th page of memory pool AB is accessed.

4.2.5 Extended addressing with data spaces

Macro	Brief description
ALESRV	Sets up and clears a connection between a program and a data space
ALINF	Provides information on the access lists with which data spaces and their connections are managed
DSPSRV	Creates, extends and deletes virtual address space for data addressing (data space), provides information on a data space and releases a data space.

The extended address space principle

The extended addressing mode is available on all BS2000 servers.

In the extended addressing mode new address spaces for data are available in addition to the address space provided in previous versions. Since some of the characteristics of these new virtual address spaces differ from those of conventional virtual address space, the following new terminology has been introduced:

– **program space** for conventional virtual address space
– **data space** for the new virtual address spaces

The following list summarizes the characteristics of a data space:
– address space for data only, i.e. addressed program code cannot be executed
– may be used in 24-bit and 31-bit addressing mode
– size of between 4K and 16Mb/2Gb, depending on addressing mode
– homogeneous address space, i.e. no memory classes, same page attributes
– contains no reserved system address areas
– may be shareable (similar to memory pools)
– data may be partitioned

Advantages of data spaces for the user:
– A considerable increase in the total volume of addressable data
– Data spaces enable the address space to be structured more easily, allowing the separation of program code and data and partitioning of data relevant to security or other critical data
– A larger program space is available to the user since user data can be stored in a data space.

Program space and data space

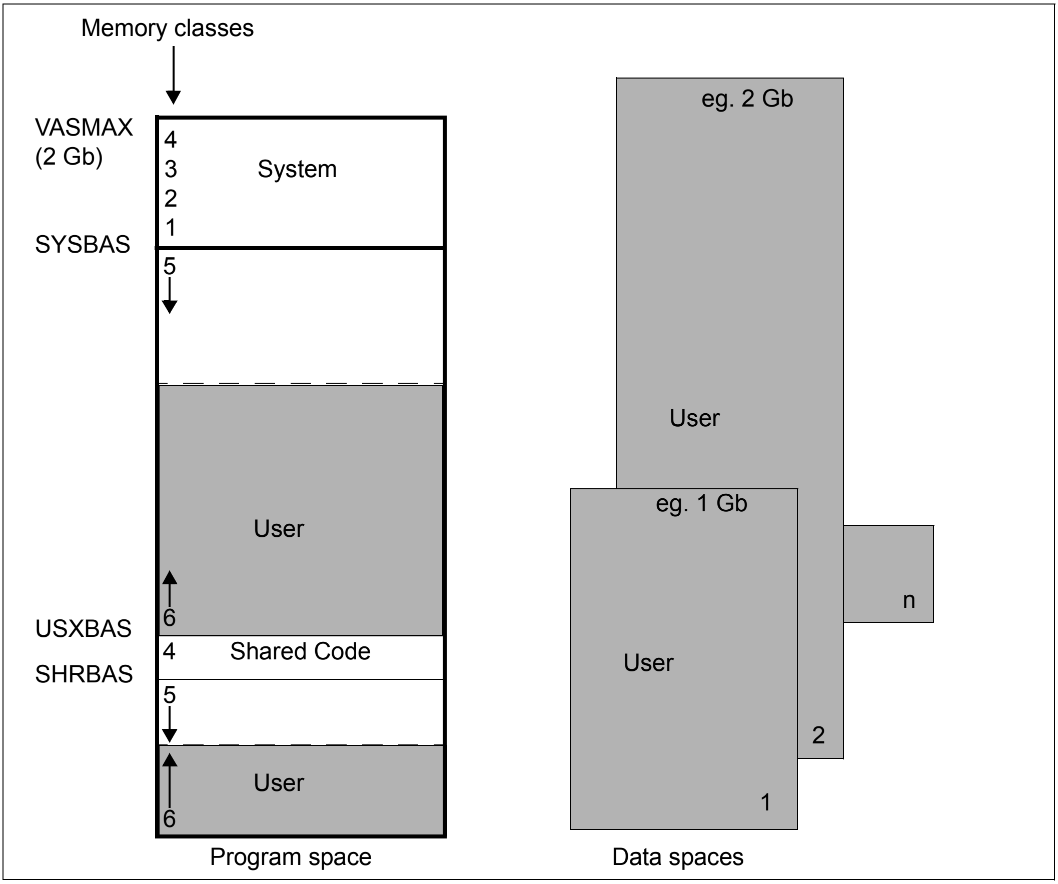


Figure 7: Extending the virtual address space with data spaces

The **program space** corresponds to the virtual address space provided in previous BS2000 versions. It therefore begins at virtual address X'00' and has a maximum size of 2Gb. Within the program space, it is possible to address both executable programs and pure data.

A **data space** is a contiguous virtual address space with a size of between 4K and 2Gb. A data space begins at virtual address 0. It is available to the user in its entirety since, unlike the program space, it does not contain any areas reserved for the system.
Within a data space, it is possible to address only data or programs that have been stored as data, i.e. program code addressed in a data space cannot be executed. A user sets up a data space by specifying type, name, scope and desired size. The user who sets up the data space becomes its owner.

A data space is implemented as a homogeneous address space, i.e. all pages are assigned the same attributes as soon as they are requested. These attributes are defined on creation of a data space. The attributes specified when a data space is created remain valid throughout the lifetime of the data space. All participants joining a data space must accept these attributes.
The scope determines which tasks can join the data space and access the data it contains. The name of a data space is unique only within its scope. The requested data space is identified throughout the session by means of the SPID (space identification). The SPID is assigned by the system.

Data space types

The data space type determines the type of memory allocation/deallocation within the data space. It is determined at generation time.

The following data space types exist:

– STACK
  A data space of the type STACK is an allocated area that is contiguous in virtual terms, starting with address 0 and going up to the current size. Functions are available for extending and reducing the current size (EXTEND/REDUCE) and for deleting the content of an area within the current size (CLEAR).

– HEAP
  A data space of the type HEAP is a virtual address space in which areas of any required size can be allocated dynamically up to the maximum size of the data space. The allocation functions available are GETAREA and RETAREA.

The size is indicated in units of 4 KB.

Addressing data space contents

Access lists

Before a program can access data in a data space, it must establish a connection to this data space.

Each task has its own access list (AL) containing all the current connections between a program and data spaces. Access lists are located in the privileged memory area and are managed by the system. When a program sets up or clears connections, entries (access list entries, ALE) are added to or deleted from the task-specific access list.

ALET and SPID

On connection setup, the program receives a value (the access list entry token or ALET) that points to the new entry in the access list.

Whereas the SPID (assigned by the system when the data space was created) is used by the software to identify the data space throughout the system, the ALET is used to address the data space on a hardware basis. The value of the ALET is task-specific:

if several tasks set up a connection to one and the same data space, they are each assigned a different ALET, since the ALET identifies an entry in the task-specific access list.

Access registers

Data spaces are accessed via an additional set of registers consisting of 16 access registers (ARs). The 16 access registers are assigned unambiguously to the 16 general registers. The general registers used for address calculation can be base or index registers. If only an index register and no base register is used for address calculation, the program space is always addressed.

If, however, a base register is (also) used for address calculation and the ALET of the corresponding access register is not zero, the corresponding data space is addressed. This makes it possible to assign a separate data space to each address that is referenced by a base register. Since general register 0 may not be used as the base register, AR0 may not be used to address a data space.

Loading the corresponding access register with the ALET that identifies a connection to a data space allows address conversion for data access to be performed via the address conversion tables of the data space.

Just as a different area within the program space can be addressed by reloading the base register with the start address of this area, a different data space can be addressed by reloading the access register with a different ALET from the access list. The access list thus represents the set of data spaces that a program can access at a particular time.

The mechanism for accessing a data space via the access list is effective only for operand access to data. In the case of branch instructions, operand access always takes place in the program space. This prevents program code in a data space from being executed.

AR mode

To make use of the extended address space option, a program must be instructed to work with the additional set of registers, i.e. to run in **AR mode** (access register mode).
If AR mode is not activated, virtual addresses of the program space are addressed.
The assembler instruction SAC activates and deactivates AR mode.

```
SAC   512      * set address space control  –  activate AR mode *
SAC   0        *                            –  deactivate AR mode *
```

If AR mode is not activated, therefore, program code and data is accessed in the program space whereas, in AR mode, data is accessed in data spaces which are indexed via access registers.
When AR mode is activated, a real address is calculated as before by converting the virtual address in two stages (index register + base register + displacement), but taking into account the access register. If the access register that corresponds to the base register ≠ 0, it refers to a data space and the virtual address of this data space is converted. If the access register = 0, a virtual address of the program space is converted.
If a program is running in AR mode and uses a general register as the base register for addressing data, the corresponding access register must be assigned a valid ALET, otherwise the program is aborted.
A special ALET value is available for addressing data in the program space. The value ALET = 0 always addresses the program space.

A number of new Assembler instructions have been introduced for working with access registers (see the “Assembler Instructions (BS2000)” manual [1]).

LAM	Load Access Multiple
STAM	STore Access Multiple
LAE	Load Address Extended
SAR	Set Access Register
CPYA	CoPY Access register
EAR	Extract Access Register
TAR	Test Access Register
SAC	Set Address space Control
IAC	Insert Address space Control

How to make a data space addressable

1. Create a data space (**DSPSRV** macro, FCT=CREATE)
   or
   supply the identification (SPID) of an existing, shareable data space (**DSPSRV** macro, FCT=INFORM).
2. Establish a connection to this data space
   (**ALESRV** macro, FCT=CONNECT). An ALET is returned.
3. Load the ALET into the access register whose corresponding general register is used as the base register for addressing the data in the data space
   (instruction **LAM AR**x,**AR**x,**alet**, if ARx EQU x is defined).
4. Activate AR mode (instruction **SAC 512**).

If more than one task is authorized to access the same data space and the data space has been made addressable via the access list, correct user serialization must be ensured.
The maximum number of data spaces that can be set up per task is 32. However, each task can access many more data spaces (already set up by other tasks). A maximum of 125 ALETs can be managed per task,i.e. a connection can be established to 125 data spaces.
All access registers are initialized with zero when the program is started.

ℹ Access registers and AR mode should be used only on a procedure-local basis, since the linkage routines are not executable in AR mode and do not save the access registers when subroutines are called.

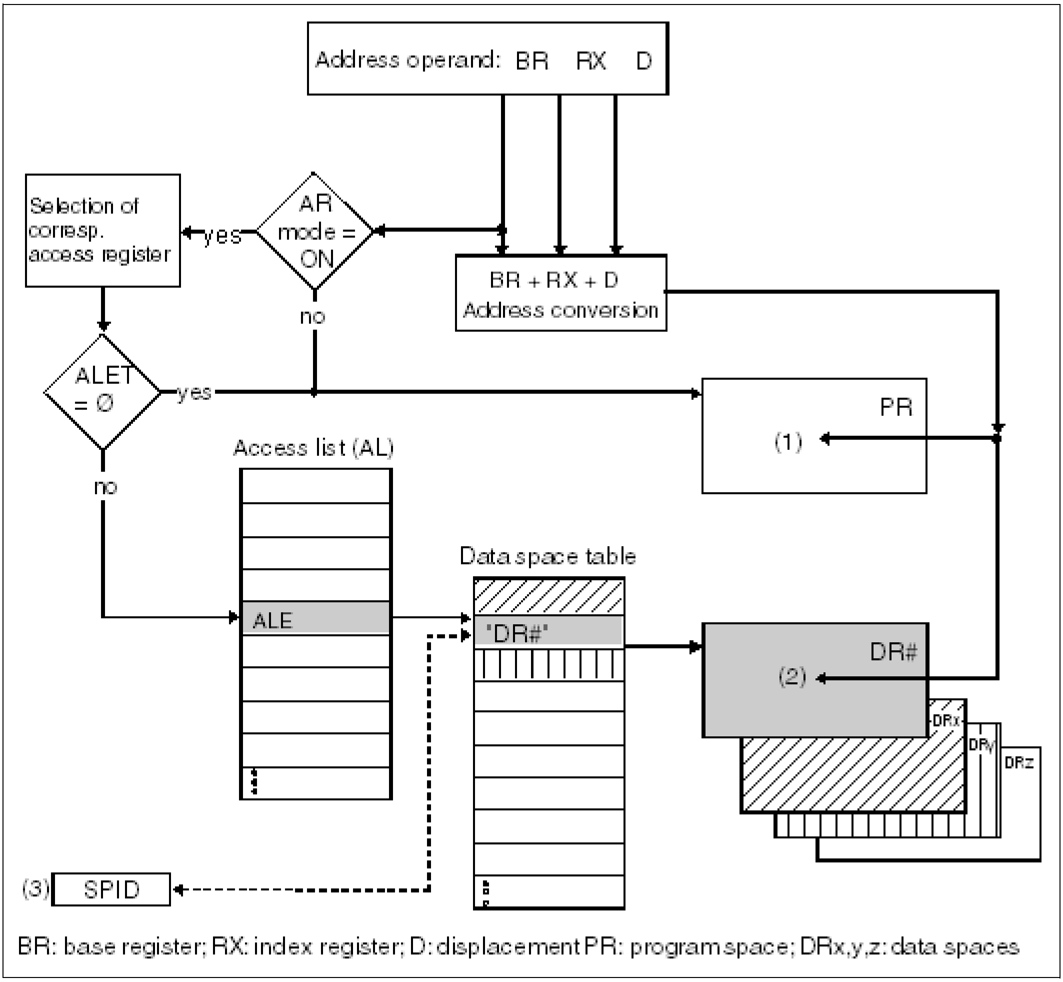


Figure 8: Selecting an address space

(1) The address calculated during address conversion refers to the program space if
- AR mode is not activated or
- the ALET in the access register (corresponding to the base register) is zero.

(2) The address calculated during address conversion refers to a data space. An access list entry (ALE) is addressed via the ALET of the corresponding access register. This ALE in turn points to an entry in the data space table via which the data space is identified.

(3) A data space can also be addressed via the SPID assigned by the system. The SPID identifies the data space throughout the session.

Macros for using data spaces

Three macros are available to nonprivileged users for working with extended virtual address space.
The **DSPSRV** macro can be used to request or release a data space. The user can request and release (additional) memory pages to an existing data space.
The **ALESRV** macro manages the entries in the access list. It connects a task to a data space and can also clear this connection. If the ALET is specified, the macro outputs the SPID associated with the data space.
The **ALINF** macro tells the user which entries in the access list refer to which data spaces.

Example

The following example illustrates the use of a type STACK data space. (@ expressions are predefined macros of the ASSEMBH assembler.) Only those sections of the full program relating to data space usage are shown.

```
DATASPAC START
         PRINT NOGEN
*        :
DSPMFD   DSPSRV MF=D
ALEMFD   ALESRV MF=D
*
DATASPAC @ENTR TYP=M
*        :
CREATE   LA    1,DSPPL ───────────────────────────────────────────── (1)
         @DATA DSECT=DSPMFD,BASE=1
         MVC   DSPPL(NVDD#),DSPMFL
         DSPSRV MF=M,FCT=CREATE,INISIZE=25
         DSPSRV MF=E,PARAM=(1)
         DSPSRV MF=R,SPID=DSPSPID
*
CONNECT  LA    1,ALEPL ───────────────────────────────────────────── (2)
         @DATA DSECT=ALEMFD,BASE=1
         MVC   ALEPL(NVDA#),ALEMFL
         ALESRV MF=M,SPID=DSPSPID
         ALESRV MF=E,PARAM=(1)
         ALESRV MF=R,ALET=DSPALET
*
         SAC   512 ───────────────────────────────────────────────── (3)
*
```

```
WRITEDS  LAM   8,8,DSPALET ——————————————————————————————————————————————   (4)
         SR    8,8
         MVC   0(100,8),DATA
*
         SAC   0 ——————————————————————————————————————————————————————————   (5)
*
INFORM   LA    1,DSPPL ——————————————————————————————————————————————————   (6)
         @DATA DSECT=DSPMFD,BASE=1
         MVC   DSPPL(NVDD#),DSPMFL
         DSPSRV MF=M,FCT=INFORM,IDENT=NAME,NAME='SHARED#DS',           C
               SCOPE=GLOBAL
         DSPSRV MF=E,PARAM=(1)
         DSPSRV MF=R,SPID=DSPSPID2
*
CONNECT2 LA    1,ALEPL ——————————————————————————————————————————————————   (7)
         @DATA DSECT=ALEMFD,BASE=1
         MVC   ALEPL(NVDA#),ALEMFL
         ALESRV MF=M,SPID=DSPSPID2
         ALESRV MF=E,PARAM=(1)
         ALESRV MF=R,ALET=DSPALET2
*
         SAC   512 ————————————————————————————————————————————————————————   (8)
*
COPYDS   LAM   7,7,DSPALET2 —————————————————————————————————————————————   (9)
         SR    7,7
         MVC   0(100,7),0(8)
*
         SAC   0 —————————————————————————————————————————————————————————   (10)
*
CLEAR    LA    1,DSPPL —————————————————————————————————————————————————   (11)
         @DATA DSECT=DSPMFD,BASE=1
         DSPSRV MF=M,FCT=CLEAR,SPID=DSPSPID2,AREA=1000,SIZE=100
         DSPSRV MF=E,PARAM=(1)
*
EXTEND   DSPSRV MF=M,FCT=EXTEND,SPID=DSPSPID,SIZE=1000 ——————————————————   (12)
         DSPSRV MF=E,PARAM=(1)
         DSPSRV MF=R,EXTADDR=DSPEXTND
*
*** Further access to the extended data space (read, write, etc.) ***
*
DISCONN  LA    1,ALEPL —————————————————————————————————————————————————   (13)
         @DATA DSECT=ALEMFD,BASE=1
         MVC   ALEPL(NVDA#),ALEMFL
         ALESRV MF=M,FCT=DISCONN,ALET=DSPALET
         ALESRV MF=E,PARAM=(1)
         ALESRV MF=M,FCT=DISCONN,ALET=DSPALET2
         ALESRV MF=E,PARAM=(1)
*
```

```
DESTROY  LA    1,DSPPL ———————————————————————————————————————————————— (14)
         @DATA DSECT=DSPMFD,BASE=1
         DSPSRV MF=M,FCT=DESTROY,SPID=DSPSPID
         DSPSRV MF=E,PARAM=(1)
         @EXIT
*
*** Definitions ***
*
DSPMFL   DSPSRV MF=L,NAME='SPACE1',MAXSIZE=2000
ALEMFL   ALESRV MF=L,FCT=CONNECT
*
DSPSPID  DS    D                   * SPID of data space created by
*                                  * program
DSPALET  DS    F                   * ALET of this data space
DSPSPID2 DS    D                   * SPID of another program's
*                                  * data space
DSPALET2 DS    F                   * ALET of this data space
DSPEXTND DS    A                   * Extension address
DSPPL    DS    XL(NVDD#)           * Dynamic data area for DSPSRV
ALEPL    DS    XL(NVDA#)           * Dynamic data area for ALESRV
DATA     DS    XL100               * Data to be transferred
         @END
```

(1) A data space with a size of 100 Kbytes is created:
The address of the parameter list for the **DSPSRV** macro is loaded into register R1 (LA), then the DSECT is placed over the parameter list (@DATA) and the parameter list is initialized (MVC). The data space is created by means of the **DSPSRV** macros. The `DSPSPID` field is to contain the SPID assigned by the system.

(2) The program connects itself to the data space:
the address of the parameter list for the **ALESRV** macro is loaded into register R1 (LA), then the DSECT is placed over the parameter list (@DATA) and the parameter list is initialized (MVC). The program connects itself to the data space via the **ALESRV** macros, specifying the SPID. The `DSPALET` field is to contain the corresponding ALET.

(3) AR mode is activated. An access register is now assigned unambiguously to each general register.

(4) Data is written to the data space:
the base register is register R8. Access register 8 is loaded with the ALET of the data space (LAM).
The base register is then deleted (SR).
100 bytes of data from the `DATA` field are written to the start of the data space (MVC).

(5) AR mode is deactivated:
AR mode should always be deactivated as soon as possible in order to avoid unintentional access to the data space (particularly when branching to subprograms).

(6) The SPID of another program's shareable data space is queried:
the **DSPSRV** macros address this data space via its name and scope. The SPID of this data space is to be entered in the `DSPSPID2` field.

(7) The program sets up a connection to the other program's data space:
the program connects itself to this data space via the **ALESRV** macros, specifying the SPID of the data space. The corresponding ALET is to be stored in the `DSPALET2` field.

(8) AR mode is activated. An access register is now assigned unambiguously to each general register.

(9) Data is copied from one data space to another:
access register 7 is loaded with the ALET of the other program's data space (LAM). 100 bytes are written to the start of the other program's data space from the program's own data space, represented by access register 8.

(10) AR mode is deactivated (see (5)).

(11) Data is deleted from the other program's data space:
the **DSPSRV** macros delete 400 Kb from this data space by overwriting them with binary zeros, starting at address X'1000'.

(12) A data space is extended:
the **DSPSRV** macros extend the program's own data space by 4000 Kb.

(13) The connection to both data spaces is cleared down:
the program clears down the connection to both data spaces via the **ALESRV** macros, specifying the appropriate ALETs.

(14) The data space created by the program is destroyed. The corresponding entry in the access list is deleted.

4.3 Task and program execution control

4.3.1 Starting, interrupting and terminating

Macro	Brief description
BKPT	Transfers control to the system. The user can then enter commands at the terminal
ENTER	Initiates a batch job (same function as ENTER-JOB command)
EXIT	Terminates an STXIT process
LGOFF	Terminates the job (same function as EXIT-JOB command)
PASS	Causes the calling program to wait one second
RETRN	Returns control to the program from which it was called (e.g. with the BR instruction). In addition it can be used to restore the contents of any registers the current program has saved using the SAVE macro
SAVE	Saves register contents by buffering them; the call is useful at the beginning of a subroutine. The register contents can be restored with the RETRN macro
SETIC	Starts or resets the interval timer (used in conjunction with the STXIT macro)
STXIT	Specifies user-defined interrupt handling routines with which the system continues processing when a program interrupt occurs
TERM	Terminates the program and the job step, and initiates a memory dump
TINF	Changes the run priority of the task, or the job type (batch, interactive or transaction), or the parameters of the task' s deactivation prohibition
VPASS	Pends the user task for a specified time

4.3.2 User and job switches

Macro	Brief description
SWITCH	Reads or changes the 32 job switches associated with the job or the 32 user switches associated with the user' s own or another user' s user ID.

The **SWITCH** macro replaces the **GETSW**, **GETUS**, **SETSW** and **SETUS** macros. These macros are still supported for reasons of compatibility only and are described in the appendix on page 1107.

Use of job switches in BS2000

When using job switches the user must bear in mind that some system components and software products alter the status of certain job switches or are controlled by them. The following table shows which software products and system components normally utilize job switches, listing them in the order in which they are subsequently described:

System component/software product	Switches
ARCHIVE	30, 31
BCAMDEF	0, 4, 5, 31
DBL/ELDE (binder/loader)	4
EDT	4 - 7
STEP/SET-JOB-STEP command	16 - 31
TSOSLNK	4

The following applies:
The loader message `BLS0500` is suppressed by setting switch 4.

After execution of the SET-JOB-STEP command, all job switches over 15 are reset.

The effect of switch usage on the system components and software products listed above is described in the following pages.

ARCHIVE

The software product ARCHIVE can be called both in procedures and in ENTER jobs. Information about program execution can be obtained from the switches set by ARCHIVE during or after execution.

<u>Switch 30 set by ARCHIVE:</u> warning message in procedures.
Switch 30 is set by ARCHIVE when the ARCHIVE statement has been executed but a warning message has been issued.

<u>Switch 31 set by ARCHIVE:</u> error in procedures.
Switch 31 is set by ARCHIVE if the ARCHIVE statement has been executed even though an error has been detected.

BCAMDEF

<u>Switches 0, 4, 5, 31 are used:</u>
Switches 0, 4, 5 and 31 are set and reset during the BCAMDEF procedure.

DBL/ELDE (binder/loader)

<u>Switch 4 set:</u> system messages concerning the loading of a module (`BLS0500`, `BLS0517`,...) are suppressed.

EDT

<u>Switch 4 set for EDT:</u> EDT suppresses start/end message

- Interactive jobs, batch jobs:
  If this switch is set before loading EDT, load message BLS0500 and, on termination of EDT, message EDT800 are suppressed. The message
  `EDITED FILE(S) NOT SAVED! TERMINATE (Y/N)?`
  is also suppressed.
- Batch jobs:
  If job switch 4 is set, no log is written during the EDT run (≙ @LOG=NONE).

<u>Switch 5 set for EDT:</u> EDT operates in L mode
Input is read line by line from SYSDTA. When output to the screen, the current line number is replaced by '*'. The @EDIT FULL SCREEN statement switches over to F mode. Activating/deactivating switch 5 during the EDT run has no effect on the mode set.

<u>Switch 6 set for EDT:</u> EDT outputs 160 characters
If this switch is set, EDT writes 160 characters in a SYSLST record and any overflow to the next record. Use of this option is recommended if the SYSLST (system) file is to be output to a printer with a maximum line length of 160 characters. The switch must be set before calling EDT.
EDT usually outputs only 132 print positions and writes any overflow to the next record.

Switch 7 set for EDT: superfluous memory space is not released
If switch 7 is set, the superfluous memory space assigned is not automatically released by EDT. EDT usually releases unoccupied memory space (negative specification in SPACE operand of FILE macro, see the “DMS Macros” manual [7]). This switch can also be set while EDT is executing.

SET-JOB-STEP command
Switches 16 through 31 are reset:
Job switches 16 through 31 are reset when the user issues a SET-JOB-STEP command.

TSOSLNK
Switch 4 set for TSOSLNK: page feed suppressed
Setting job switch 4 causes all page feeds in linkage editor listings to be ignored on output to SYSLST.

4.3.3 Intertask communication (ITC)

Macro	Brief description
CLCOM	Terminates participation in intertask communication
OPCOM	Opens intertask communication for a participant, and supplies an ITC name
RELBF	Deletes the first message in the ITC receive queue of the ITC participant
REVNT	Receives a message for the ITC name of the participant
SEVNT	Sends a message to another ITC participant

Intertask communication (ITC) procedure

Intertask communication (ITC) allows message exchanges between programs which are being executed in different tasks. ITC requires each ITC participant to have a unique identity in the form of an ITC name. From the viewpoint of the operating system, ITC participants are those tasks which have an ITC name maintained in the communication table; from the user's viewpoint, they are the (user) programs which are being executed in these tasks and which have called **OPCOM**. By participating in intertask communication, all tasks in the system can exchange messages and - if necessary - wait for their arrival, provided they know each other's ITC names. Each ITC participant may send and receive messages. A participant (program) can send a message to another participant, but will not be informed automatically of receipt of the message. The receiver can acknowledge receipt by returning a message.

In addition to sending messages, a participant may ask to be given messages already sent or yet to be sent by other participants. If the message is not yet available at the time of the request, the program run can be halted for a selected period of time. As soon as the message arrives, or on expiration of the waiting period, the system will resume the program run and deliver the message. The receiver can specify the participant whose message is to terminate this waiting period, or can accept any participant as the sender.

The following macros are used for intertask communication:

OPCOM	Open ITC participation
SEVNT	Send message
REVNT	Request message
RELBF	Delete receive queue
CLCOM	Close ITC participation

Opening and closing ITC participation

A program wanting to participate in ITC must call the **OPCOM** macro, and give an ITC name. The system checks this name and rejects it if it has already been assigned to another ITC participant. If the name is unique and if sufficient system memory (for ITC participant lists) is available, the system includes the task with its ITC name in the participant list. If the task is the first ITC participant, the **OPCOM** call also initiates the intertask communication (the system sets up a communication table). **OPCOM** is rejected if insufficient system memory is available for the lists and tables.

As soon as the **OPCOM** call has been accepted by the system, the participant can send messages to or receive messages from other ITC participants, itprovided that their ITC names are known to one another.

The user can terminate participation in ITC by means of the macro **CLCOM**. If the NOKEEP operand is specified in the macro, participation is terminated completely for that user, and no further messages may be sent or received. The system deletes any existing messages in the receive queue and erases the user's ITC name from the participant list.

If the user calls the **CLCOM** macro with the KEEP operand, then only the participation as receiver is terminated. The system delivers no messages, but the user can still evaluate the receive queue and send messages to other participants. If the receive queue is empty when the **CLCOM** KEEP call is issued, the system proceeds as for the **CLCOM** NOKEEP call.

It is recommended that ITC participation be terminated with the call sequence **CLCOM** KEEP/**CLCOM** NOKEEP, in order to prevent messages being lost. Otherwise, messages arriving between the **REVNT** and the **CLCOM** calls will be lost.

If a program or task is terminated without the macro **CLCOM** having been called, the system will terminate ITC participation for this program or task with an internal call of the **CLCOM** NOKEEP macro. This may result in messages being lost. To avoid this, the receive queue can be evaluated in an STXIT routine (see page 131). The routine should issue the call **CLCOM** KEEP, check any messages present, return the proper acknowledgment and close with **CLCOM** NOKEEP, thus forestalling system-internal program/task termination (including **CLCOM** NOKEEP).

The receive queue

For each ITC participant, the system sets up a receive queue in system memory, in which it queues all messages to the participant in the order of their arrival. The participant must evaluate and delete the messages in the queue.

If a participant requests a message (**REVNT** macro), the system passes to it the message that came in first, and which is also the first in the queue (FIFO principle). If the participant requests a message from a specific sender, it receives the first message transmitted by this sender. The system always transfers the message that is first in line. Thus, a participant wanting to request the next message must first delete the first one. When a message is initially requested (with a **REVNT** macro), the system can be directed to delete it from the queue after it has been transferred. Depending on the particular request, this may be the first message in the queue or the first one from a specific sender. The first message from a specific sender cannot be deleted explicitly, but only in conjunction with the request (**REVNT**). The first message in the queue can, however, be deleted explicitly with the **RELBF** macro. The participant can delete the whole receive queue by calling the **RELBF** macro in a loop and testing the return code each time to check whether the receive queue is empty.

As soon as participation in ITC is terminated (with **CLCOM**), the system stops accepting messages for the receive queue of the participant concerned. Simultaneously with the closing of ITC participation, the program or task can have the system cancel its receive queue. Alternatively, it can continue to evaluate the existing messages and subsequently release the receive queue by means of a further **CLCOM** call.

Sending messages

An ITC participant can send messages to any other participant at any time (using the **SEVNT** macro), provided sufficient system memory is available. The sender merely needs to know the ITC name of the receiver. The system does not, however, notify the intended receiver that a message has arrived. The addressed participant must take the initiative and request the message. Once the message has been received, it is good practice for the receiver to send back a message as an acknowledgment. The sender can then request this at a convenient time (e.g. after all messages have been sent off) by means of a **REVNT** macro.

The sending participant holds the message ready in a separate program area. The message may be up to 64 K in length. After calling the **SEVNT** macro, the system checks whether the receiver's ITC name is syntactically correct, and whether it is contained in the participant list. It also makes sure that the messages contained in the receive queue do not exceed a total length of > 128 K. If the check proves positive, the system transfers the message to the receiver's receive queue. The sending task is continued without waiting after the **SEVNT** call and can send further messages to any desired receiver.

Requesting and receiving messages

Each ITC participant can request a message with the **REVNT** macro and - if it has not yet arrived - wait for it. The receiver can specify that the message is to come from a specific sender or that a message from any participant will be accepted. If the receive queue is empty or contains no messages from the desired sender, the system interrupts the task after the **REVNT** call until the message arrives or until the waiting time has expired (see figure 9 on page 80). The waiting time interval (between 1 second and 6 hours) can be set in the **REVNT** call. As soon as the task is continued, the **REVNT** return code can be tested to check whether the message is arrived.

The request for a message (**REVNT** macro) can be linked with eventing. The eventing mechanism regards the arrival of the message as an ITC event (see following section).

The system transfers the message from the receive queue into a program area of the participant, together with the sender's ITC name and its length. If the receiving field is not large enough to accommodate the complete message, the system transfers only the header, consisting of the name of the sender, the length of the complete message and the first 4 bytes of the message.

A participant requesting a message with **REVNT** can also specify that the system is to delete it from the receive queue after transmission (operand REL=YES). Operand REL=NO causes the message to be retained. This allows the same message to be requested more than once. Until the first message in the queue is deleted it will continue to be the one transferred with each request. To have the next message transferred, the participant must delete the first one (implicitly with **REVNT** or explicitly with **RELBF**).

If a participant receives messages of different lengths, it may not be advisable to reserve a full-length destination field (64K + 7 for sender name and message). If **REVNT** is called with a length of 16 (and REL=NO), only the message header (sender name, record length field and 4 bytes of information) will be transmitted. The participant can then decide either to delete the message or to request more memory space (**REQM**) to enlarge the destination field. A further request can then be made for the message, this time in its full length.

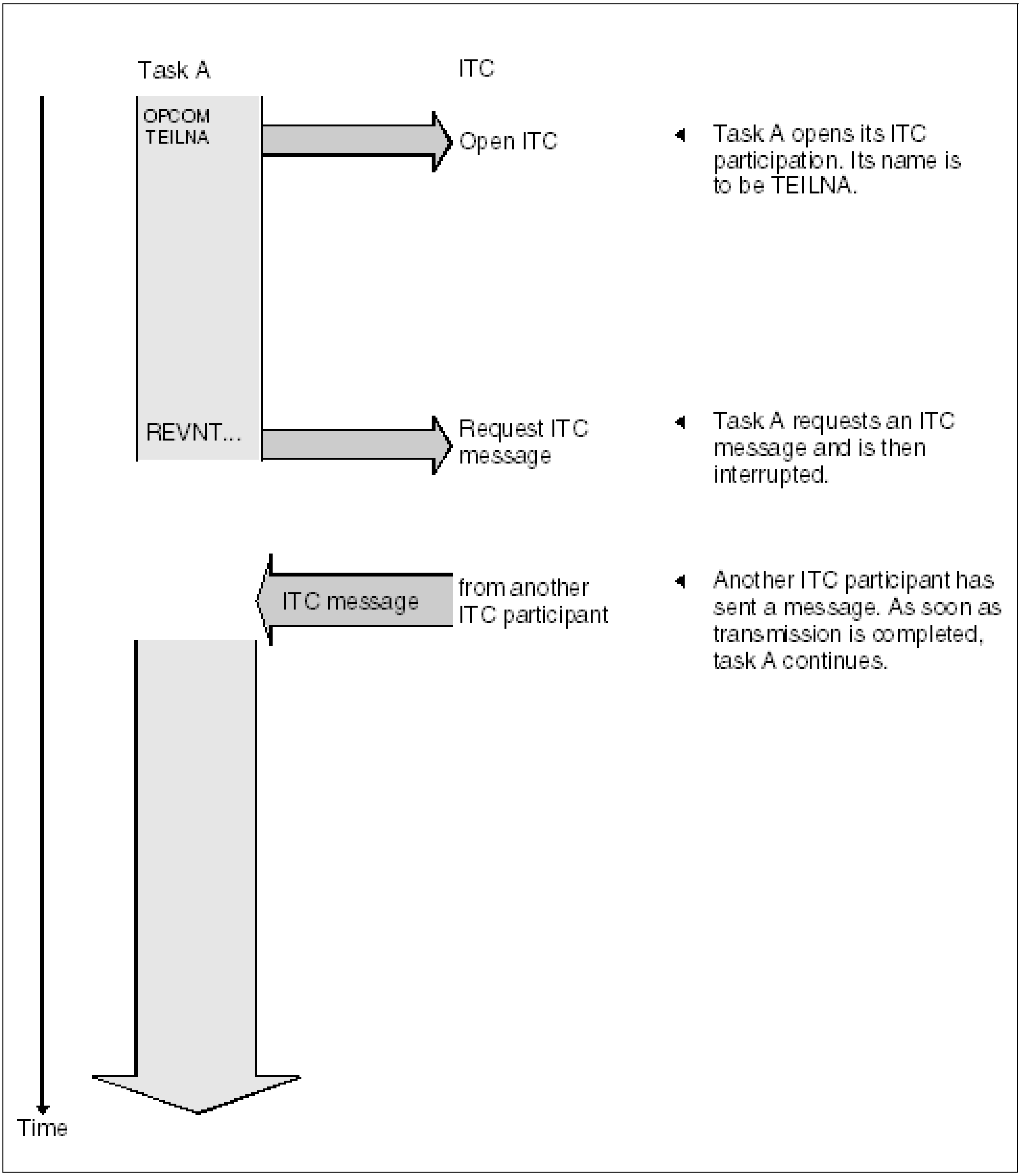

Task A
ITC
OPCOM
TEILNA
Open ITC
Task A opens its ITC participation. Its name is to be TEILNA.
REVNT...
Request ITC message
Task A requests an ITC message and is then interrupted.
ITC message
from another ITC participant
Another ITC participant has sent a message. As soon as transmission is completed, task A continues.
Time

Figure 9: Intertask communication (ITC): program requesting a message

Linking ITC to eventing

This section assumes that the reader is familiar with eventing, which is described on page 94 and page 110 of the present manual.

Event classes:

Eventing permits users to have the execution of their program interrupted until one of a number of different events occurs (see page 94). The events that can influence eventing are subdivided into various classes according to their origin (see figure 10 on page 82). At present the following event classes are available:

Optional events
UPAM events
ITC events
DCAM events
CJC events

In the class of optional events, the user determines when and whether to signal an event to eventing (**POSSIG**, see page 103). Events of all other classes (ITC, UPAM etc.), though originating during program execution, are signaled to eventing by the system, not by the program (internal **POSSIG**). In addition, the nature of such events is defined:
for example, a UPAM event occurs upon completion of an I/O operation, and the arrival of an ITC message (or expiration of a specified waiting time) is regarded as an ITC event.

A program which solicits a signal from an event item (**SOLSIG**) may also be the one in which the expected event occurs. For example, a program initiates an I/O operation via PAM and then requests a signal for the event (**SOLSIG**). As a result, the program is interrupted (synchronous eventing) by the event item and continued as soon as an internal **POSSIG** indicates that the I/O operation has been completed.

A participant cannot restrict its signal request to a specific event class. The signal received (continuation or contingency start) may refer to any event that may occur within the scope of the event item. By means of the post code (see page 96), eventing informs the participant of the class and precise circumstances of the event.

Execution of the program can therefore be made dependent on a number of different events. This is useful when several events are expected but the program does not know which of them will occur first (e.g. arrival of ITC message or reading of a PAM block completed).

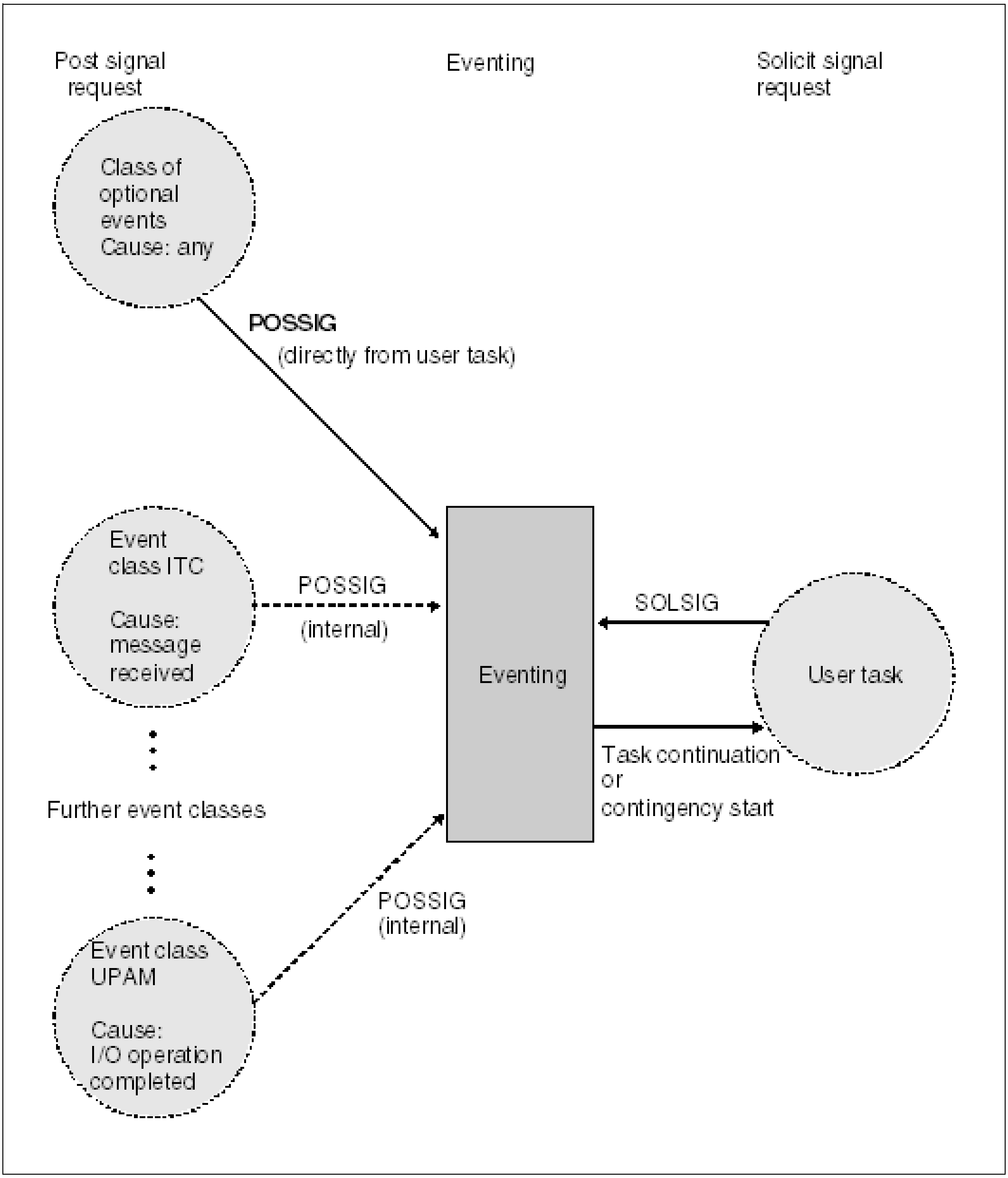


Figure 10: Event classes

ITC message as an ITC class event

The effect of linking ITC and eventing is that an (ITC) event occurs whenever an ITC message is received or the waiting time expires. The link is established through specification of an event item ID address in the **REVNT** call. As a result, program execution will not be interrupted by the **REVNT** call. The message sender need not be a participant in eventing.

Linking ITC and eventing permits more flexible program execution than ITC alone:

- The participant can combine a wait for an ITC message with a wait for another event.

  Example

  The participant program issues a **PAM** (see “DMS Macros” manual [7]) and a **REVNT** call, both linked to an event item. It then issues a **SOLSIG** call and is interrupted. Upon its resumption, it checks the post code to see whether the message has arrived, or the I/O operation has been completed.

- The participant can postpone the wait interrupt: The interrupt occurs not after the **REVNT** call, but after the subsequent **SOLSIG** call (see figure 11 on page 86 and figure 12 on page 87).

  Example

  Using a linked **REVNT** call, the participant requests an ITC message and continues processing. At some later point it issues a **SOLSIG** call to check whether the message has arrived. Only if the message is still outstanding will the interrupt occur.

- The participant can avoid any wait interrupt by specifying a contingency routine in the **SOLSIG** call (see figure 13 on page 88). In such an asynchronous instance of eventing, the **SOLSIG** call may equally well be issued prior to the **REVNT** call.

  Example

  To evaluate an ITC message and to perform all actions contingent on it, a contingency routine is defined in a program. The contingency routine is specified in the **SOLSIG** call. The message is requested by a **REVNT** call given before or after the **SOLSIG** call. Program execution is not interrupted either by **REVNT** or by **SOLSIG** (which would cause it to wait), but only by the contingency routine, which starts its execution as soon as an event occurs.

The **SOLSIG** call does not enable a participant to request a specific event. Eventing assigns to the participant the first event occurring within its scope which has not yet been requested by other (previous) **SOLSIG** calls. By means of the post code, the participant must filter out the relevant events.
The scope of an event item may comprise one task, all tasks under one user ID, or all tasks within the system. The scope is defined when enabling the event item.
Several event items (with various scopes) can exist at the same time, and a task may associate itself with various event items at the same time (see page 94).

Program structure for the linked use of ITC and eventing (without contingency process) (see figures 11 on page 86 and 12 on page 87).

- Participation in both ITC and eventing must be enabled (**OPCOM** and **ENAEI**). The system enters the ID of the event item under the address specified in the **ENAEI** macro. If the event item has already been enabled, the participant has to belong to the specified scope in order to join in. If other participants within the same scope also call the **SOLSIG** macro, it is impossible to predict to which participant an event will be assigned by the event item.
- A message is requested with the **REVNT** macro. In addition to the known operands, the address of the ID used in the **ENAEI** call is specified in the EIID operand.
- Execution of the program is not interrupted. The program has to check the **REVNT** return code in R15 to ascertain whether the call has been accepted, or rejected due to a format error. A rejected **REVNT** call does not produce an ITC event. The return code for a linked **REVNT** cannot contain any details of the message; only the post code contains such details.
  No further **REVNT** call (linked or unlinked) may be issued prior to the completion of a linked **REVNT** call (i.e. message received or waiting time expired). Other calls linked to the event item, e.g. a PAM call, are permissible.
- At a time convenient for its execution, the program calls the **SOLSIG** macro and specifies an address for the post code. Program execution is not interrupted if the signal request is immediately satisfied. Otherwise, in the case of synchronous eventing, program execution is interrupted for a period not longer than the specified waiting time interval.
- Upon continuing its execution, the program first checks the return code of the **SOLSIG** call for format errors or expiration of the SOLSIG waiting time. (The expiration of the **REVNT** waiting time is specified in the post code.)
- The post code is checked. The system has stored it at the address specified in the **SOLSIG** call (or in register 3 for a contingency process). The leftmost byte of the post code indicates the class to which a specific event belongs (ITC event: X'08'). The program then proceeds to the processing section provided for the particular event class.

- Processing of an ITC event: The rightmost byte of the post code is checked. It indicates whether the **REVNT** waiting time has expired or whether the message, or only the message header, has been transmitted. From this point on, **REVNT** calls may again be issued.

  For many ITC applications, one message is likely to be immediately followed by others. In such a case, any additional **REVNT** calls should not be linked to eventing. The **SOLSIG** calls required are time-consuming, and are of no advantage if the message has already arrived. Instead, the program will continue to issue unlinked **REVNT** calls specifying WTIME=0 until all outstanding messages have been received. Subsequently, a linked **REVNT** call will again be useful for expecting further messages.

- The program terminates its participation in ITC and eventing by means of **CLCOM** and **DISEI** (see note).

Note: terminating event-driven ITC

Until an event has been signaled for a linked **REVNT** call (i.e. message arrived or waiting time expired) the participant may not disable the associated event item. Otherwise, the contingency process will be started due to the **DISEI** call (asynchronous eventing), or, with synchronous eventing, an ITC event can no longer be signaled even if the message has already been transmitted. The following procedure is recommended: The participant must check internally whether the last linked **REVNT** has been completed (e.g. by means of a counter; for local scope, with **CHKEI**). The **DISEI** call can then be issued. By issuing **CLCOM** KEEP, the participant can suppress the reception of any further messages. Any messages still waiting in the message queue are picked up by means of unlinked **REVNT** calls. Afterwards, ITC participation can be terminated with **CLCOM** NOKEEP.

Note: waiting times

Both the **REVNT** call and the **SOLSIG** call can be used to set a waiting time.
The REVNT waiting time limits the duration of the message request. If the waiting time expires and no message has arrived, this is regarded as an ITC event. The system issues an internal **POSSIG** call to the event item, which in turn signals an ITC event to the participant. Upon expiration of the REVNT waiting time, the rightmost byte of the post code contains the value X'10'.
The SOLSIG waiting time limits the duration of the solicit signal request. If the waiting time expires and the event item has not signaled an event, the interrupted program is continued (synchronous operation) or the contingency routine is started (asynchronous operation). Upon expiration of the SOLSIG waiting time, the return code of the **SOLSIG** call has the value X'20000004'. (Expiration of the REVNT waiting time is regarded as an ITC event. In this case, the **SOLSIG** return code has the value X'00000000').

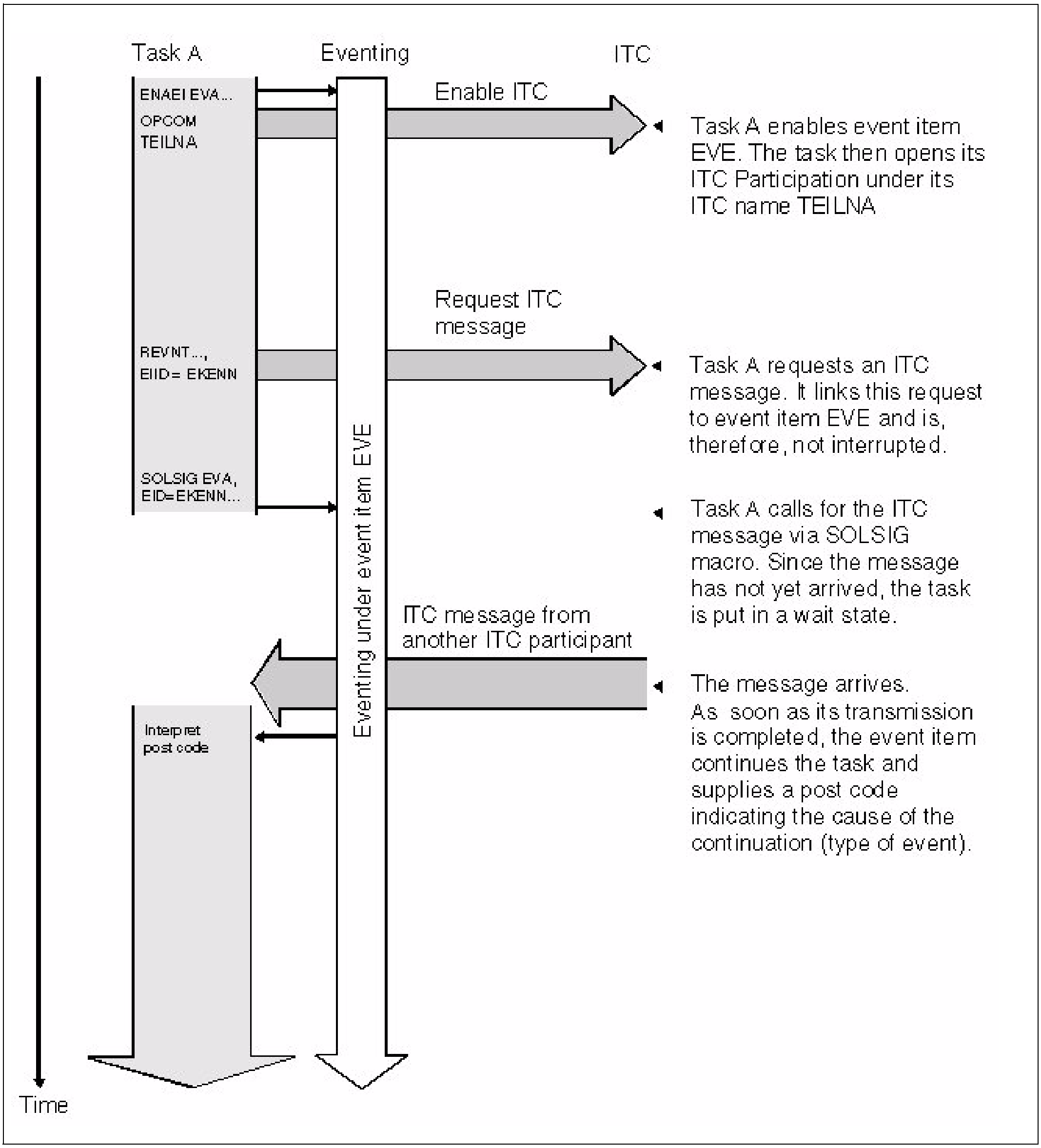
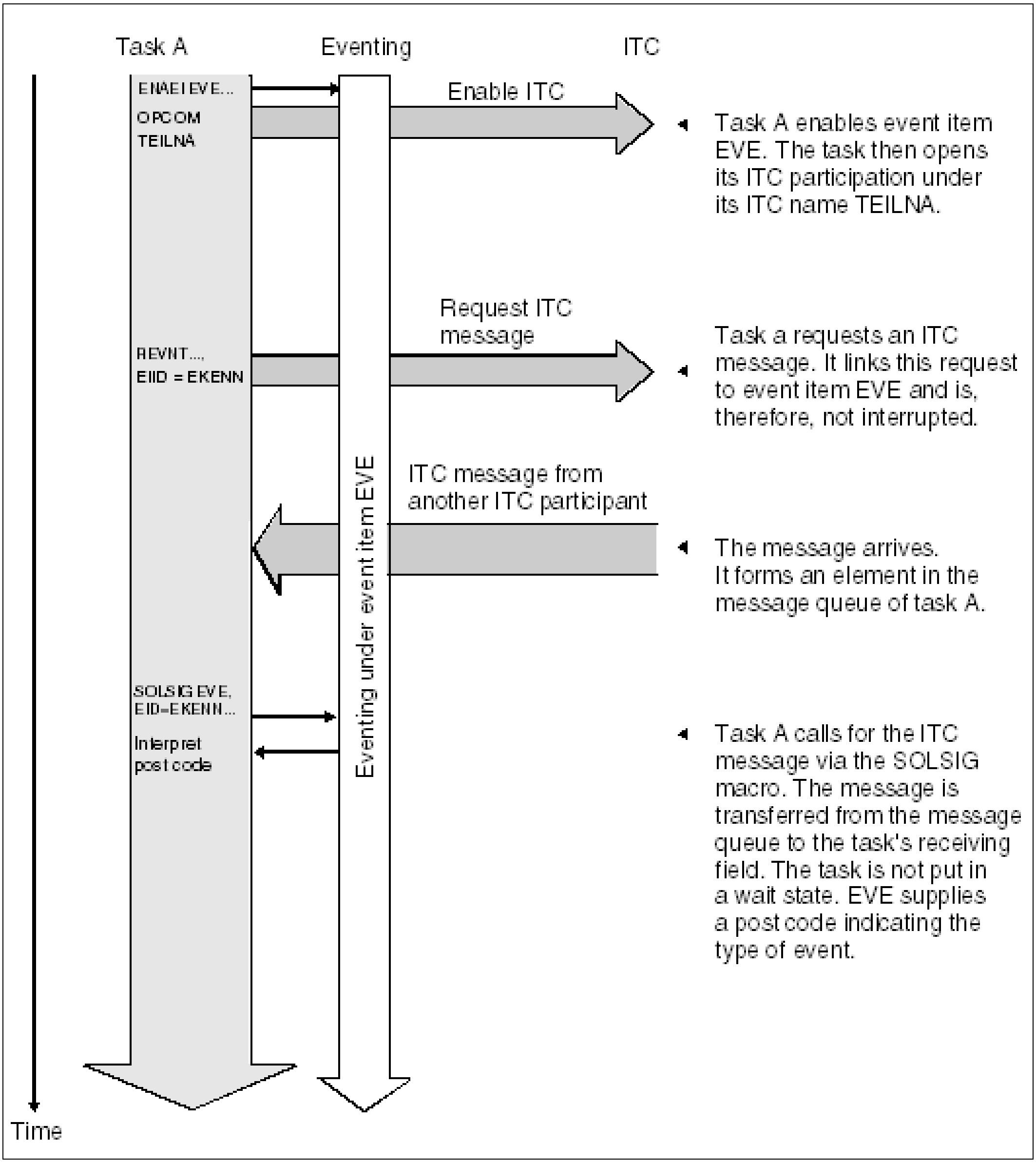

Task A
Eventing
ITC
ENAEI EVA...
OPCOM
TEILNA
Enable ITC
Task A enables event item EVE. The task then opens its ITC Participation under its ITC name TEILNA
Request ITC message
REVNT...,
EIID= EKENN
Task A requests an ITC message. It links this request to event item EVE and is, therefore, not interrupted.
SOLSIG EVA,
EID=EKENN...
Task A calls for the ITC message via SOLSIG macro. Since the message has not yet arrived, the task is put in a wait state.
Eventing under event item EVE
ITC message from another ITC participant
The message arrives. As soon as its transmission is completed, the event item continues the task and supplies a post code indicating the cause of the continuation (type of event).
Interpret post code
Time

Figure 11: ITC linked to eventing (synchronous operation):
The requested message arrives after the SOLSIG call



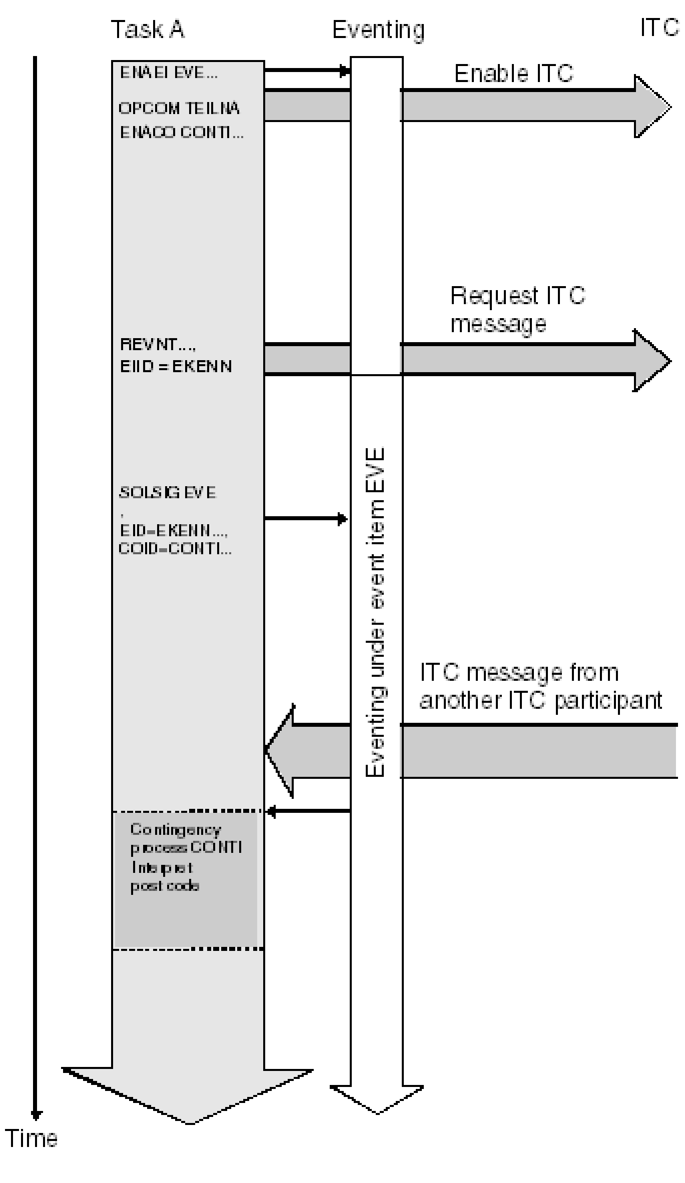
Task A enables event item EVE. The task then opens its ITC participation under its ITC name TEILNA.

Task a requests an ITC message. It links this request to event item EVE and is, therefore, not interrupted.

The message arrives. It forms an element in the message queue of task A.

Task A calls for the ITC message via the SOLSIG macro. The message is transferred from the message queue to the task's receiving field. The task is not put in a wait state. EVE supplies a post code indicating the type of event.

Figure 12: ITC linked to eventing (synchronous operation):
The requested message arrives before the SOLSIG call



Task A enables event item EVE (ENAEI). The task then opens its ITC participation (OPCOM). It also defines contingency process CONTI.

Task A requests an ITC message. It links this request to event item EVE.

Task A calls for the ITC message via SOLSIG and, at the same time, specifies contingency process CONTI, which is to be initiated as soon as the message has arrived.

The message arrives. As soon as its transmission is completed, the event item initiates contingency process CONTI and supplies a post code indicating the type of event.

Figure 13: ITC linked to eventing (asynchronous operation):
A contingency process is executed when the message arrives.

Example

Program structure for the linked use of ITC and eventing, with a contingency process (see figure 13 on page 88).

```
EXAMPLE  START
         BALR   6,0
         USING  *,6
         :
         :
         ENAEI  EINAME=EVE,EIIDRET=ECODE ..............  (1)
         ENACO  CONAME=CONTEVE,COADAD=CONTADR,
                COIDRET=CONTI
         OPCOM  PARTIC
         :
         :
         SOLSIG EIID=ECODE,COID=CONTI,LIFETIM=3600 ...  (2)    Program
         C      15, RC00                                       segment with ITC
         BNE    SOLSERR                                        processing (basic
         :                                                     process)
         :
         REVNT  RECEIPT,100,WTIME=600,EIID=ECODE ......  (3)
         C      15,RC00
         BNE    REVERR
         :
         :
         CLCOM ........................................  (4)
         DISCO  COID=CONTI
         DISEI  EIID=ECODE
         TERM

CONTANF  BALR   5,0 ...................................  (5)
         USING  *,5
         ST     3,POSTCODE
         :                    Checking of post codes for
         :                    various event classes
         CLI    POSTCODE,ITCEVENT                              Contingency
         BE     ITCPROC                                        routine
         :                    Processing of events belonging   (contingency
         :                    to other classes                 process)
         :
ITCPROC  :                    Checking of post codes for
         :                    reception of messages.
         :                    Evaluation of the message
         RETCO
```

```
REVERR   (Error handling in REVNT call)
         :
         B      ....
         :                                                      Error routine
SOLSERR  (Error handling in SOLSIG call)
         :
         B      ....
         :
ECODE    DS     F
CONTADR  DC     A(CONTANF)
CONTI    DS     F
RECEIPT  DS     CL100                                           Data
RC00     DC     A(0)
POSTCODE DC     A(0)
ITCEVENT EQU    X'08'
         END
```

(1) Participation in eventing is enabled. The name of the event item is EVE. The address of the event item ID is ECODE. The scope is local.
A contingency routine is defined. Its ID resides at the address CONTI. ITC participation is opened. The ITC name is PARTIC.

(2) Using the **SOLSIG** call, an event signal is requested. A contingency routine is specified in the call. This is to be started if a message arrives or, at the latest, upon expiration of the waiting time. The **SOLSIG** call may also be issued after the **REVNT** call.
The basic process continues (unless interrupted by the contingency process because the message had already arrived). The **SOLSIG** return code is checked. If it is not equal to zero, the system did not accept the call. The SOLSERR error routine checks to find the cause.

(3) An ITC message is requested. The specification EIID=ECODE links this request to event item EVE. Execution of the program is not interrupted. Calls for other event classes (e.g. PAM) may be issued before or after this **REVNT** call.
A return code in R15 not equal to zero indicates that the call was not accepted. The REVERR error routine checks to find the cause.

(4) The participant program wants to terminate linked ITC processing: It terminates participation in ITC with **CLCOM**, deletes the contingency definition with **DISCO** and disables event item EVE with **DISEI**. (For ways of preventing messages being lost, see the note preceding the example.)

(5) The contingency routine stores the post code supplied by the system in register R3 and then evaluates its leftmost byte. This byte indicates the class of the event that initiated the contingency routine. In the processing section for the corresponding event class, the contingency routine evaluates the rightmost post code byte, which

contains the return code for the class. For ITC events, this return code indicates whether the waiting time has expired or whether the message has been transmitted (see **REVNT**). Then if, for example, the waiting time has expired, it can issue another linked **REVNT** call (a return to the first **REVNT** call is not possible). The contingency routine is terminated by the **RETCO** macro.

4.3.4 (Task) serialization

Macro	**Brief description**
CHKSI	Checks the queue occupancy of a serialization item
DEQAR	Terminates the (exclusive) use of a serialization item by the task associated with the calling program
DISSI	Cancels the assignment of the task to the serialization item. The serialization item is deleted if it is not being used by any other task
ENASI	Assigns the task to a serialization item. The serialization item is established if it is not already being used by another task
ENQAR	Requests the (exclusive) use of a serialization item. The request is entered in a queue

Introduction

The user is supplied with a semaphore-type mechanism which permits serial access to certain items (serialization items).

Serialization item

A serialization item is not linked with any particular characteristic. It is the responsibility of the user to confer a particular characteristic and identity on the serialization item.
A serialization item is identified by a name; an ID is also provided and can be used in further macros of the same task in order to speed up processing.

Scope of a serialization item (SCOPE operand)

The SCOPE operand specifies the scope (participating group) for a serialization item. The scope may encompass a single task, all the tasks under one ID, or all the tasks in the system.

Enabling the use of a serialization item

A serialization item must be assigned to a task before an (exclusive) access request for the item (**ENQAR**) can be processed.
A serialization item is assigned to a task either explicitly by means of **ENASI** (ENAble Serialization Item) or implicitly using **ENQAR** (ENQueue Access Request) with a name. If there is already a serialization item with the stated name in the defined scope (established by an **ENASI** macro in another task), the macro merely causes the serialization item to be assigned to the task associated with the calling program. Otherwise the serialization item is established and assigned by the system. An ID is provided for the serialization item only if an explicit enable function (**ENASI**) is used.
A task can use up to 2000 serialization items simultaneously.

Canceling the assignment of a serialization item

A serialization item assignment is canceled either explicitly by means of a **DISSI** macro (DISable Serialization Item) or implicitly by means of a **DEQAR** macro (DEQueue Access Request) with the **DISSI** operand. If the serialization item is not used by any other task, it is deleted.

An **ENASI** macro for a serialization item that has already been disabled does not necessarily produce the same ID assignment as that established by the preceding **ENASI** call.

Program termination

With program termination, the assignment of the task to all the serialization items used is canceled.

Access request for a serialization item (ENQAR macro)

The **ENQAR** macro (ENQueue Access Request) requests (exclusive) access to the specified serialization item. The request is entered in the serialization item queue and the task is placed in a wait state until it is first in the queue, when it resumes execution and uses the serialization item until a **DEQAR** call (DEQueue Access Request) is issued for this item.

If no serialization item with the specified name exists in the defined scope, one is established and assigned (implicit enable function).

Using the COND operand, the user can specify whether the access request is to be satisfied immediately or whether it may wait. The waiting time is specified by the LIFETIM operand.

Terminating access to a serialization item (DEQAR macro)

The **DEQAR** macro (DEQueue Access Request) terminates a task's access to the specified serialization item.

Access can be terminated by the task which requested the access or by any other task which also uses this serialization item (operand HOLDER=ANY in the **DEQAR** call).

Optionally (DISSI operand), the assignment between the task and the serialization item can also be canceled (implicit disable function).

Checking a serialization item (CHKSI macro)

The **CHKSI** macro (CHecK Serialization Item) provides information about availability and access (for the task associated with the calling program or any other task) of the specified serialization item. This information is made available as a return code (in register R15).

4.3.5 Eventing

Macro	Brief description
CHKEI	Checks the queue status for an specified event item
DELFEI	Deletes a SOLSIG/POSSIG entry in the EVENTLST (optimized eventing, forward eventing)
DISEI	Terminates participation in eventing. The specified event item is deleted if it is not being used by any other task
DPOFEI	Generates a POSSIG entry in the EVENTLST (optimized eventing, forward eventing)
DSOFEI	Generates a SOLSIG entry in the EVENTLST (optimized eventing, forward eventing)
ENAEI	Enables participation in eventing. The task is assigned to the specified event item
POSSIG	Signals the occurrence of an event to a specified event item
RPOFEI	Signals an event to the event item (optimized eventing, forward eventing)
RSOFEI	Requests notification of the arrival of a signal from an event item (optimized eventing, forward eventing)
SOLSIG	Issues a solicit a signal request for an event item

General

Eventing is a technique which makes it possible to coordinate the execution of two or more (user) programs in different tasks. For example, program B is to read data records from a file only after program A has updated it; or program A is to write a data record in a shared memory area only after program B has read the preceding record. Coordination of the programs is achieved by the use (for control purposes) of a common event variable to determine the actions of the operating system (to put a task into a wait state, to end the wait state, or to start a contingency process). The event variable is addressed via the event item. The entire procedure - posting of events, their interrogation and the consequent actions taken - is referred to as eventing. The external representation of eventing is the event item.

From the viewpoint of the operating system, the participants in eventing are the tasks which are associated with a common event variable (event item); from the user's viewpoint, the participants are the (user) programs which are executed in these tasks.

Basic functions of eventing

- Enabling participation in eventing
  Programs whose execution is to be coordinated must open (enable) their participation in eventing and specify the name (and scope) of a common variable (event item). The declarations concerning participation, and the specification of the event item, apply only to the program currently being executed.
  For each event item, two queues are set up (POSSIG and SOLSIG queues). A task may use up to 2000 event items simultaneously.

- Sending a signal
  A program wishing to report the completion of a particular processing operation sends a POSSIG signal to eventing. This signal is entered in the POSSIG queue (on the FIFO principle).

- Requesting a signal
  A program requiring information about the completion of a particular processing step in another program sends a SOLSIG request to eventing. This request is entered in the SOLSIG queue (on the FIFO or LIFO principle, according to specification).

- Servicing the queues (see figure 18 on page 102)
  The first POSSIG signal in the POSSIG queue is assigned to the first SOLSIG request in the SOLSIG queue - irrespective of which tasks they originated from. If the SOLSIG request cannot be satisfied (i.e. the POSSIG queue is empty), two courses of action are possible:

  – synchronous eventing (see figure 15 on page 99 and figure 16 on page 100)
    The task which issued the SOLSIG request is put into a wait state either until a POSSIG signal is received by eventing or until the specified waiting time has elapsed. (Execution of the program is synchronized using the POSSIG signal.)

  – asynchronous eventing (see figure 17 on page 101)
    The task which issued the SOLSIG request is not put into a wait state. If the SOLSIG request can be satisfied during the subsequent continued running of the program, or a long enough time elapses to satisfy the specified waiting time, then a contingency routine specified in the program is started.

  If the SOLSIG request can be satisfied immediately (the POSSIG queue is not empty), then either the program execution will continue with the instruction which follows the **SOLSIG** macro, or a contingency routine specified in the **SOLSIG** call is started.
  Each participant can check the status of the queues; the information given indicates whether there are any POSSIG signals in the POSSIG queue, and whether there are any SOLSIG requests in the SOLSIG queue.

- Disabling eventing
  Participation in eventing can be disabled at any time. The participant is deleted from the list of participating tasks. Any SOLSIG requests which are still in the SOLSIG queue at that time are deleted. Any POSSIG signals not yet assigned at that time remain in the POSSIG queue.
  Participation is implicitly terminated when a program ends (**not** at the end of the task). The event item and all the internal processing lists are deleted when the last (or only) program terminates its participation.

- Post code (see figure 18 on page 102)
  Eventing only registers the arrival of a POSSIG signal. No information is provided about the event giving rise to the signal (file closed, record written, ...). Since the first entry in one queue is always paired with the first entry in the other queue, it is not possible to address a particular receiver or sender directly. However, the sender of the POSSIG signal may send a short message (post code) with the signal. The code is passed to the task whose SOLSIG request is satisfied by this POSSIG signal. It enables the receiver to determine whether the POSSIG signal is connected with its program execution or not.

The **post code** is 4 or 8 bytes long (4 bytes when using the 24-bit interface). The table below summarizes the important applications for various classes of event:

Application/ Event class	**Post Code**	**Remarks**
(POSSIG)	X'aa....aa'	aa....aa: string specified by the user (4 or 8 bytes)
ITC	X'08000000' X'08000004' X'0800000C' X'08000010'	REVNT event terminated Operand error Receiving field too small Timeout
DCAM	X'0Caa...a'	aa...a: character string provided by the user (3 or 7 bytes)
UPAM	X'1000i..a'	31-bit addressing mode i..a = iiiiaaaaaaaa, where: iiii: identifier which the user assigned to the job (2 bytes) a...a: PAM data area address (4 bytes) The user may employ either the data area address or just the first word plus the identifier iiii
	X'10aaaaaa'	24-bit addressing mode a...a: PAM data area address (3 bytes)
CJC	X'1400iiii' X'1404iiii' X'1408iiii'	Condition satisfied Job variable deleted Catalog exported } iiii: Character string to identify the ONEVT

Table 7: Applications/event classes and their post codes

The macro **ETCNAM** (with no operands) is used to generate symbolic names for the individual event classes.

```
           ETCNAM
1                *,MACRO: ETCNAM, VERSION: VER040
1 *
1 *                                 EVENT TYPE CODES WHICH MAY BE
1 *                                 OUTPUT FROM THE SYSTEM TO THE USER
1 *
1 ETCTCS   EQU   X'04'              TCS-EVENT
1 ETCITC   EQU   X'08'              ITC-EVENT
1 ETCDCM   EQU   X'0C'              DCAM-EVENT
1 ETCUPM   EQU   X'10'              UPAM-EVENT
1 ETCCJC   EQU   X'14'              CONDITIONAL JOB CONTROL-EVENT
```

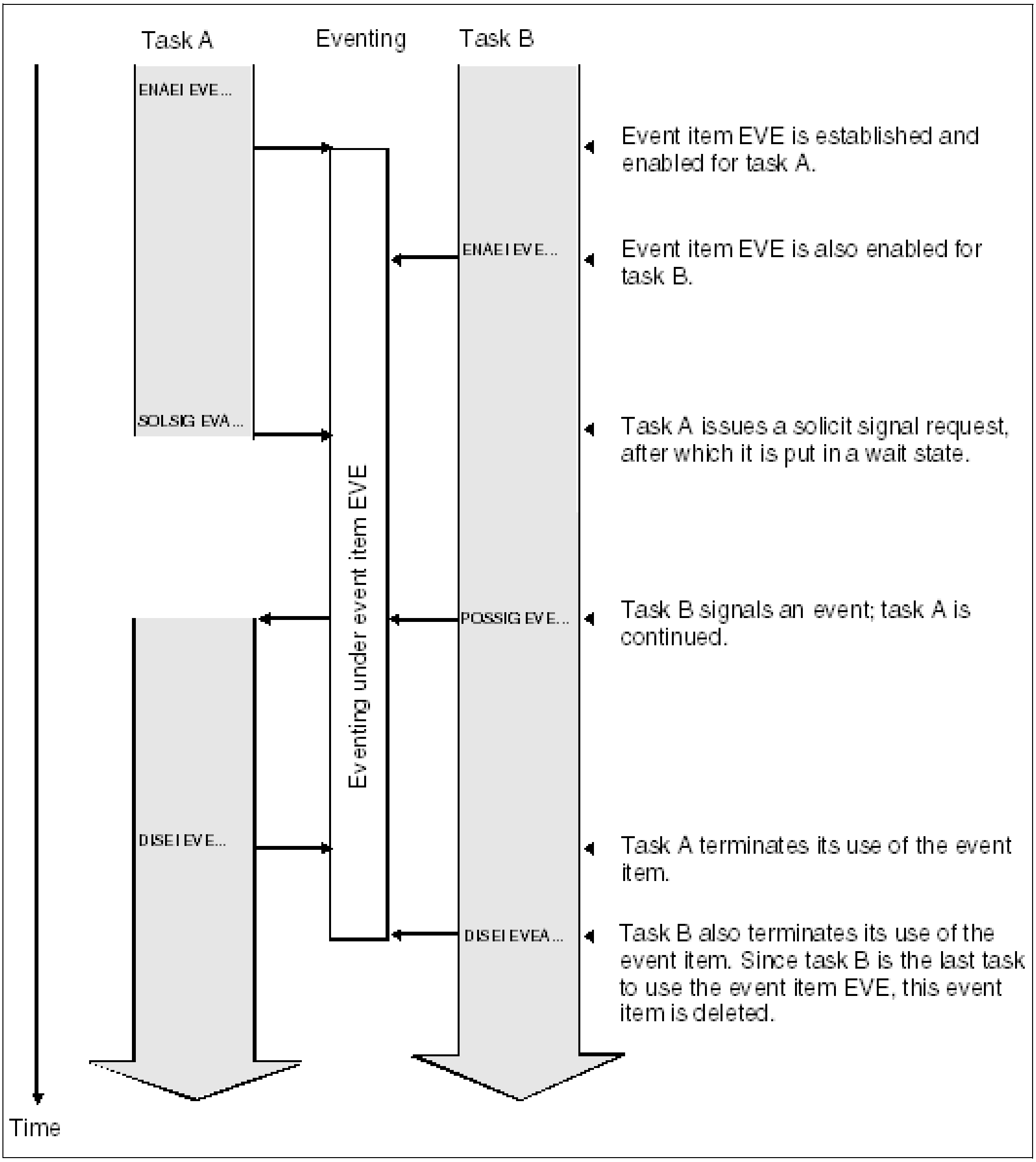

Task A
Eventing
Task B
ENAEI EVE...
ENAEI EVE...
SOLSIG EVA...
POSSIG EVE...
DISEI EVE...
DISEI EVEA...
Eventing under event item EVE
Time
Event item EVE is established and enabled for task A.
Event item EVE is also enabled for task B.
Task A issues a solicit signal request, after which it is put in a wait state.
Task B signals an event; task A is continued.
Task A terminates its use of the event item.
Task B also terminates its use of the event item. Since task B is the last task to use the event item EVE, this event item is deleted.

Figure 14: Eventing: Synchronous operation
The event is signaled before the waiting time has expired

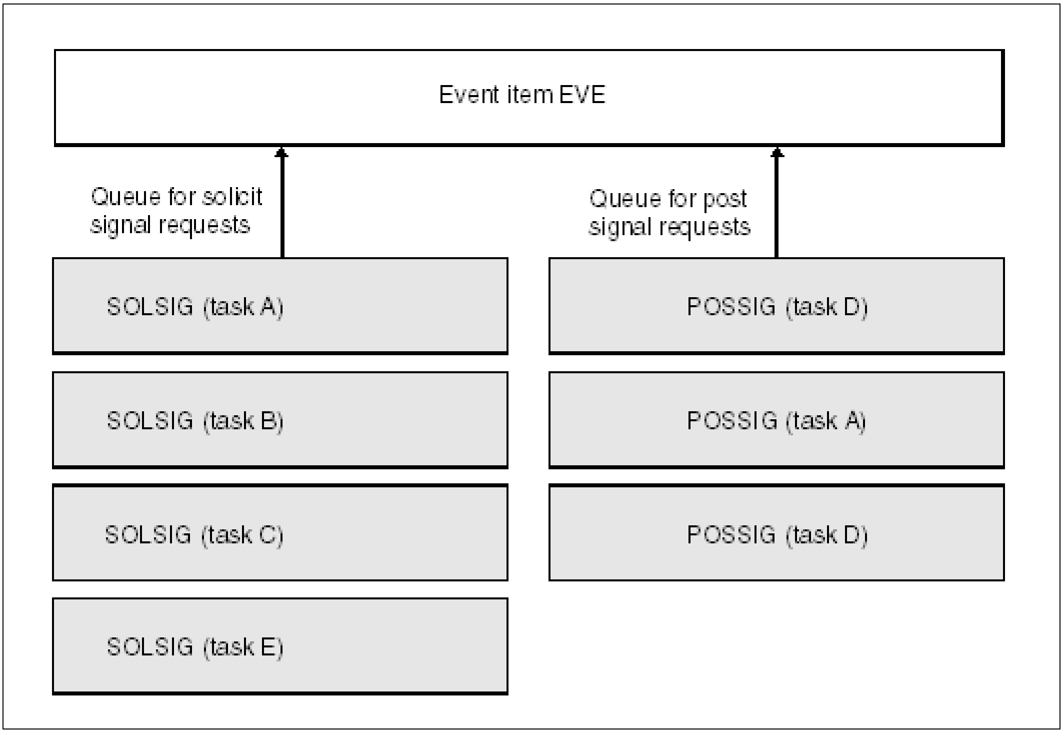
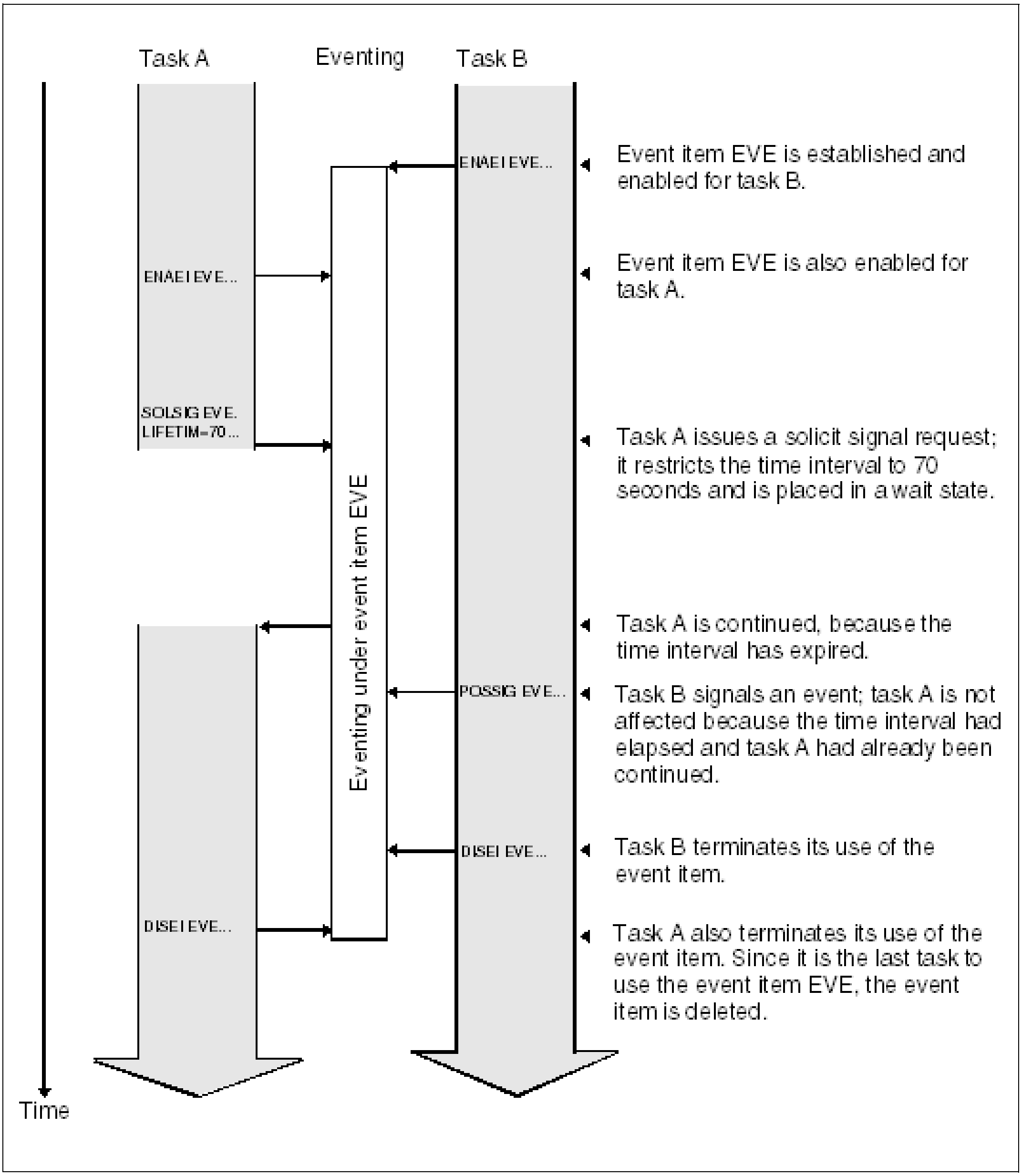


Figure 15: Event item queues



Event item EVE is established and enabled for task B.

Event item EVE is also enabled for task A.

Task A issues a solicit signal request; it restricts the time interval to 70 seconds and is placed in a wait state.

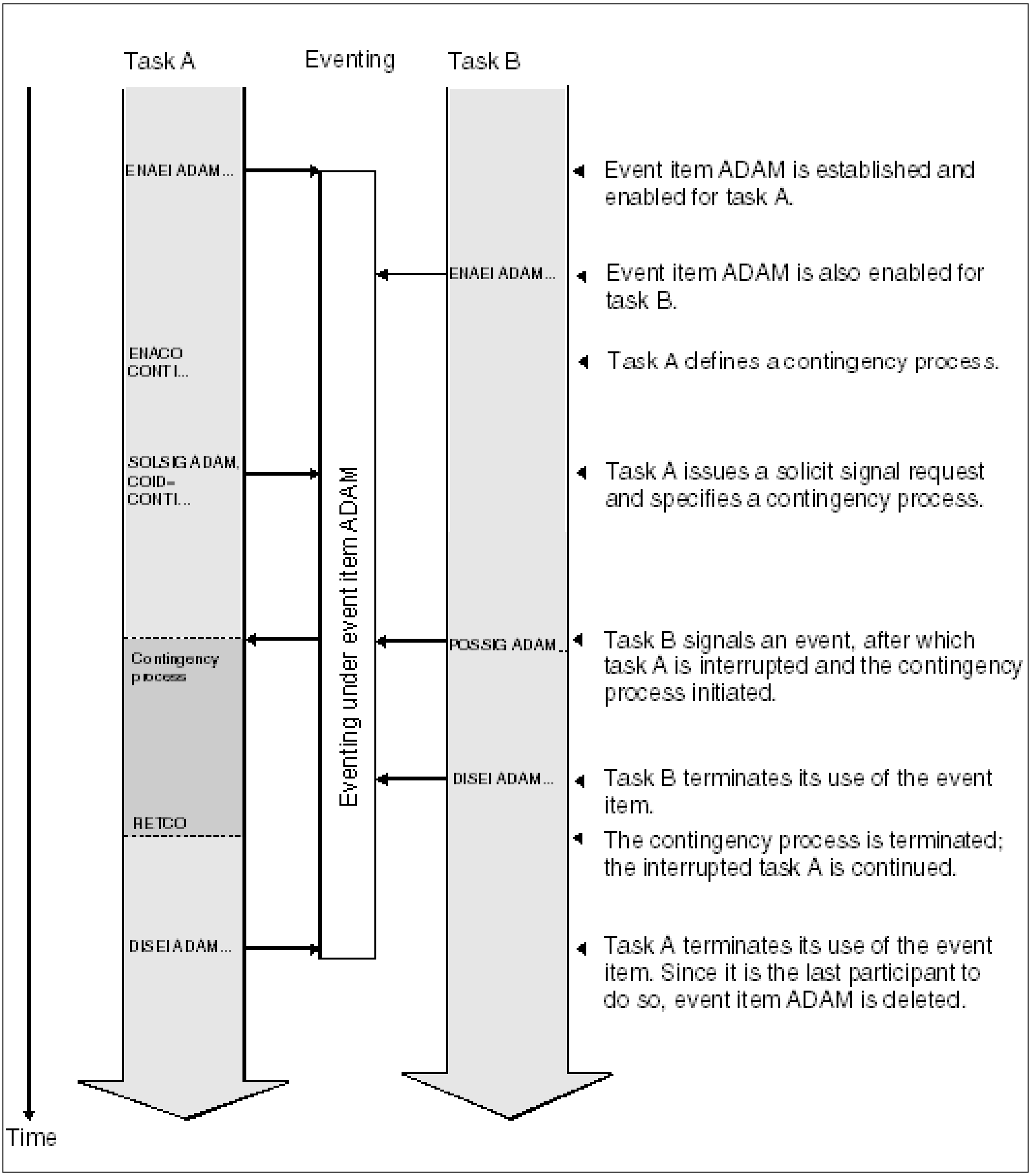
Task A is continued, because the time interval has expired.

Task B signals an event; task A is not affected because the time interval had elapsed and task A had already been continued.

Task B terminates its use of the event item.

Task A also terminates its use of the event item. Since it is the last task to use the event item EVE, the event item is deleted.

Figure 16: Eventing: Synchronous operation
The event is signaled after the waiting time has expired



Event item ADAM is established and enabled for task A.

Event item ADAM is also enabled for task B.

Task A defines a contingency process.

Task A issues a solicit signal request and specifies a contingency process.

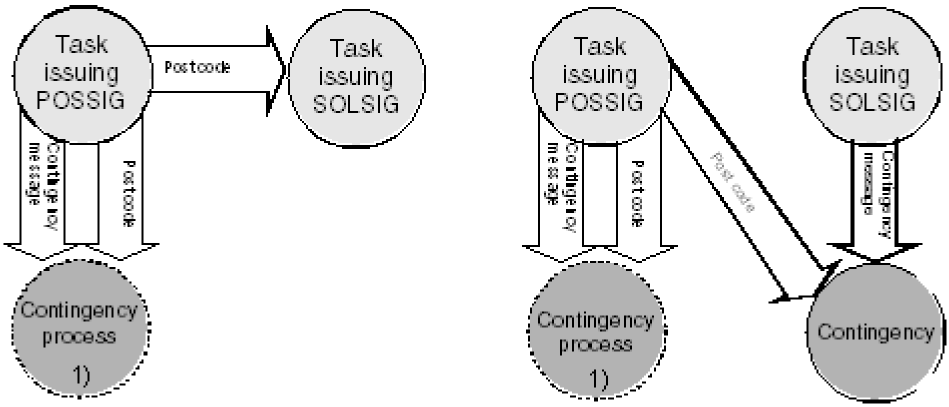
Task B signals an event, after which task A is interrupted and the contingency process initiated.

Task B terminates its use of the event item.

The contingency process is terminated; the interrupted task A is continued.

Task A terminates its use of the event item. Since it is the last participant to do so, event item ADAM is deleted.

Figure 17: Eventing: Asynchronous operation
The event is signaled before the waiting time has expired



1) the process that issues the POSSIG job can optional specify a dependent process

Figure 18: Information transfer (eventing)

The basic functions described are implemented using two groups of macros. The **user eventing** group of macros allows comprehensive use of the basic functions. The **forward eventing** (FEV) group of macros supplements user eventing with optimized variants which make for more efficient execution. Forward eventing can only be used in the synchronous mode. User eventing and forward eventing macros can be used together.

User eventing

User eventing is implemented using the following macros:

ENAEI	Enable participation in eventing
POSSIG	Send a POSSIG signal
SOLSIG	Register a SOLSIG request
CHKEI	Check event item queue
DISEI	Disable participation in eventing

- Participation in eventing is enabled by calling the macro **ENAEI**. The macro call must include a name for the event item and the scope (group of participants in eventing). The name is only valid within the specified scope. These two items of information are used internally to construct an ID for the event item. The ID is passed back to the calling participant and can be used in subsequent eventing macros (this speeds execution of the macros). Each participant in eventing receives a unique ID. If a user disables eventing in a program, and then later re-enables it in the same program run, the ID which is then issued may be different from the first one. Eventing is enabled by the first participant to issue the **ENAEI** macro. The event item and the requisite internal processing lists are then established automatically; subsequent participants are assigned to the event item.
  The scope may include a task, all tasks under a user ID, or all tasks in the system (the same name associated with a different scope would represent a different eventing procedure).

- The **POSSIG** macro is used to send a signal to eventing. The sending of the POSSIG signal does not cause the task to be interrupted. The macro call may specify a contingency routine that is to be started either if it was possible to assign a SOLSIG request to the POSSIG signal, or if a specified time allowed for such an assignment has expired (the SOLSIG queue is empty). After the waiting time has expired, the POSSIG signal is deleted from the queue.
  When it is executed, the contingency routine supplies an event information code in register R2 (see page 115). The event information code indicates whether the expected event has occurred (a POSSIG signal was assigned) or not; the code thus represents a kind of acknowledgment of the POSSIG signal.

  The **POSSIG** call may also specify a contingency message, which will be passed to the contingency routine in register R1 after it has started.
  Several consecutive **POSSIG** calls may be chained together. The chain may also terminate with the macro **SOLSIG**.

- The **SOLSIG** macro is used to send a SOLSIG request to eventing. In the case of synchronous eventing, the task is put into a wait state until a POSSIG signal can be assigned to the SOLSIG request or until the specified waiting time has expired. It is also possible to specify that the task should not wait for the arrival of the POSSIG signal. In the case of asynchronous operation, the task is not put into a wait state. A contingency routine specified in the macro call is started if either

  a) a POSSIG signal can be assigned to the SOLSIG request within a specified waiting time, or
  b) the waiting time has expired.

  In situation a) it is possible to direct that a new SOLSIG request be sent immediately.

  The following items of information are passed on to the contingency process:

  – the contingency message specified in the **SOLSIG** macro in register R1 of the contingency process.
  – the event information code in register R2.
  – the post code from the **POSSIG** macro in register R3 (plus register R4).

  In both synchronous and asynchronous operation, the SOLSIG request is deleted from the SOLSIG queue on expiry of the waiting time.

- The macro **CHKEI** can be used to check the occupancy of the queues. The information requested is supplied to the participant in the form of the return code.

- The macro **DISEI** terminates participation in eventing. Any SOLSIG requests remaining in the SOLSIG queue are deleted. Assigned forward events are also deleted (see below). Any POSSIG signals not yet assigned remain in the POSSIG queue. The event item is deleted when the last (only) participant issues the **DISEI** macro.

Forward eventing

Forward eventing supplements user eventing with the following macros:

DPOFEI	Create a POSSIG entry in the EVENTLST
DSOFEI	Create a SOLSIG entry in the EVENTLST
RPOFEI	Send a POSSIG signal
RSOFEI	Register a SOLSIG request
DELFEI	Delete a POSSIG-/SOLSIG entry in the EVENTLST

Forward eventing (FEV) is an optimized form of synchronous eventing. FEV avoids the need for repeated validation of the operands when either **POSSIG** or **SOLSIG** calls to a particular event item are repeated. Instead, an event list (EVENTLST) is set up and in the case of SOLSIG requests, for example, a SOLSIG entry is made. In subsequent steps in the program, any (real) SOLSIG request will simply refer to this entry. The entry can be explicitly deleted subsequently. The same applies to the sending of POSSIG signals. For each participant, a maximum of 2047 entries can be created in the EVENTLST. Participation in eventing must be enabled using the macro **ENAEI** and terminated using **DISEI**. The macro **CHKEI** can be used to check the POSSIG or SOLSIG queue. From the viewpoint of eventing, it makes no difference whether, for example, a POSSIG signal is sent using **POSSIG** or using **RPOFEI**. The same applies to the sending of a SOLSIG request. Forward eventing can only be used in synchronous operation. A post code may be specified.

- The macro **DPOFEI** creates a POSSIG entry in the EVENTLST. A reference number for the entry is returned to the caller. As in the case of the **POSSIG** macro, several successive **DPOFEI** calls can be chained. The specifications given in the macro call are copied into the EVENTLST entry. A post code may be specified. However, a contingency routine (**POSSIG** macro) must not be specified.
- The macro **DSOFEI** creates a SOLSIG entry in the EVENTLST. A reference number for the entry is returned to the caller. The operand values specified in the macro call are copied into the EVENTLST entry. Only synchronous operation is permitted.
- An **RPOFEI** macro referencing the POSSIG entry in the EVENTLST causes a POSSIG signal to be sent to eventing.
- An **RSOFEI** macro referencing the SOLSIG entry in the EVENTLST causes a SOLSIG request to be registered with eventing.
- The **DELFEI** macro can be used to delete a POSSIG entry or a SOLSIG entry in the EVENTLST.

Example: synchronous operation

The programs EV1 and EV2 are started in separate interactive jobs (at different data display terminals). EV2 is reading from a file that EV1 wants to copy. EV2 signals the closing of the file to EV1; then EV1 copies the file.
Both programs are to run in 31-bit addressing mode: EV1 below and EV2 above the 16-Mb boundary.

Program EV1

```
EV1      START
EV1      AMODE ANY
EV1      RMODE ANY
         GPARMOD 31
         PRINT NOGEN
         BALR  3,0
         USING *,3
         ENAEI EINAME=EVE,SCOPE=GROUP,EIIDRET=ABRID1 ——————————————————  (1)
         GDATE TOD=TIME1
         WROUT MESS1,ERROR
         SOLSIG EIID=ABRID1,COND=UNCOND,RPOSTAD=PCEMPF,RPOSTL=2,       -
               LIFETIM=800 ———————————————————————————————————————————  (2)
M1       GDATE TOD=TIME2
         WROUT MESS2,ERROR ———————————————————————————————————————————  (3)
         CMD   'COPY-FILE','TEST.FILE.1,TEST.FILE.2'
         WROUT TEXT,ERROR
         DISEI EIID=ABRID1 ———————————————————————————————————————————  (4)
         TERM
ERROR    TERM  DUMP=Y
***      DEFINITIONS     ***
ABRID1   DS    F
MESS1    DC    Y(MESS1END-MESS1)
         DS    CL2
         DC    X'01'
         DC    C'PROGRAM WAITING SINCE '
TIME1    DS    CL8
MESS1END EQU   *
MESS2    DC    Y(MESS2END-MESS2)
         DS    CL2
         DC    X'01'
         DC    C'POSSIG SIGNAL RECEIVED AT '
TIME2    DS    CL8
         DC    C'; POSTCODE = '
PCEMPF   DS    CL8
MESS2END EQU   *
```

```
TEXT     DC    Y(TEXTEND-TEXT)
         DS    CL2
         DC    X'01'
         DC    C'COPY-FILE COMMAND EXECUTED'
TEXTEND  EQU   *
         END
```

(1) Event item EVE is enabled.

(2) EV1 issues a **SOLSIG** call for EVE and provides a receiving field for a post code (2 words). Then EV1 waits for a maximum of 800 seconds for the arrival of a POSSIG signal.

(3) After the arrival of the POSSIG signal (or after the wait time has finished), EV1 is continued and outputs a message. EV2 has closed the TEST.FILE.1 file; EV1 can copy the file.

(4) EV1 disables event item EVE.

Program SRC.EV2

```
EV2      START
EV2      AMODE ANY
EV2      RMODE ANY
         GPARMOD 31
         PRINT NOGEN
         BALR  3,0
         USING *,3
         ENAEI EINAME=EVE,SCOPE=GROUP,EIIDRET=ABRID2 ——————————————————   (5)
         OPEN  TESTFCB,INPUT ——————————————————————————————————————————   (6)
         GET   TESTFCB,INAREA
CLOSE    CLOSE ALL
         GDATE TOD=TIME1
         POSSIG EIID=ABRID2,SPOSTAD=PCSEND,SPOSTL=2 ———————————————————   (7)
         WROUT MESS1,ERROR
         DISEI EIID=ABRID2 ————————————————————————————————————————————   (8)
         TERM
ERROR    TERM  DUMP=Y
***      FILE     ***
         DS    0F
TESTFCB  FCB   FCBTYPE=SAM,LINK=FILIN,EXIT=E1
E1       EXLST EOFADDR=CLOSE,COMMON=CLOSE
***      DEFINITIONS     ***
ABRID2   DS    F
PCSEND   DC    X'C5E5F26060C5E5F140'  FIELD ALIGNED ON WORD BOUNDARY!
```

```
MESS1    DC   Y(MESS1END-MESS1)
         DS   CL2
         DC   X'01'
         DC   C'POSSIG SIGNAL SENT AT '
TIME1    DS   CL8
MESS1END EQU  *
INAREA   DS   CL200
         END
```

(5) EV2 is connected to event item EVE.

(6) EV2 opens the TEST.FILE.1 file and reads a data record. The file is closed.

(7) EV2 sends a POSSIG signal and transfers a post code (2 words).

(8) EV2 disables event item EVE.

Runtime log of the interactive job with program EV1.SRC

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,ev1), -
//         compiler-action=module-generation(module-format=llm), -
//         module-library=macexmp.lib, -
//         listing=parameters(output=*library-element(macexmp.lib,ev1))
%  ASS6011 ASSEMBLY TIME: 525 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 86 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=ev1 ———  (1)
%  BLS0523 ELEMENT 'EV1', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'EV1', VERSION ' ' OF '<date> <time>' LOADED
PROGRAM WAITING SINCE 13:14:41 ————————————————————————————————————————  (2)
POSSIG SIGNAL RECEIVED AT 13:20:22; POSTCODE = EV2--EV1
COPY-FILE COMMAND EXECUTED ————————————————————————————————————————————  (3)
```

(1) EV1 is loaded and started.

(2) EV1 has been waiting for the POSSIG signal.

(3) The POSSIG signal was received; EV1 copies the file.

Runtime log of the interactive job with program EV2.SRC

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,ev2), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,ev2))
%  ASS6011 ASSEMBLY TIME: 871 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 86 MSEC
//end
%  ASS6012 END OF ASSEMBH
/add-file-link link-name=filin,file-name=test.file.1
/start-executable-program library=macexmp.lib,element-or-symbol=ev2 ───  (4)
%  BLS0523 ELEMENT 'EV2', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'EV2', VERSION ' ' OF '<date> <time>' LOADED
POSSIG SIGNAL SENT AT 13:20:22 ────────────────────────────────────────  (5)
```

(4) EV2 is loaded and started.

(5) EV2 sends the POSSIG signal, after it has closed the TEST.FILE.1 file.

For further **examples** see the section on contingency processes (page 110) and the **POSSIG** and **SOLSIG** macro descriptions.

4.3.6 Contingency processes

Macro	Brief description
CONTXT	Reads or writes in the registers of the interrupted contingency process or of the basic process
DISCO	Disables the definition of a routine as a contingency process
ENACO	Enables a routine as a contingency process and assigns a contingency name and a priority to it
LEVCO	Changes the priority of the calling basic process or contingency process during execution
RETCO	Terminates the calling contingency process. The system continues the process with the basic process or another contingency process
SUSPEND	Suspends the calling basic process or contingency process in an interruptible state

Use of contingency processes

Contingency processes are used in conjunction with eventing and STXIT procedures.
The use of contingency processes in STXIT procedures is a special application and results in certain restrictions with regard to the following description; these are dealt with in “STXIT procedure with contingency processing”, page 131.
They consist of user-written routines that are processed by the system as contingency processes.

A contingency process has the following characteristics which distinguish it from a task on the one hand and from a routine on the other:

- A contingency process has no task sequence number (TSN) of its own.
- Contingency processes can be nested.
- A contingency process is not started by the user issuing SET-LOGON-PARAMETERS; it is initiated by the occurrence of a user-defined event.
- A contingency process has its own processing level (priority), its own process control block (PCB) and thus its own set of registers.

A contingency process, which by definition is initiated when an event occurs, enables the user to take measures that are specifically related to this event. An example of such an event could be that another process has reached a particular stage during processing, such as the generation of a file or the start of a program.

The measures required to handle several events can be precisely coordinated by the assignment of appropriate priorities.

At its start, the registers of the contingency process contain the value zero, and a base register must be defined and loaded. When selecting the base register, note that registers R1, R2, R3 and possibly R4 are used by the system to transfer information to the contingency process.

Contingency processes can be employed by the user by issuing the following macros:

ENACO	Enable contingency definition
DISCO	Disable contingency definition
RETCO	Return from a contingency process
CONTXT	Access context of a process
LEVCO	Change priority level of a process

Enabling and disabling a contingency definition

A contingency process is defined by means of the **ENACO** macro, in which name, start address and priority are specified. The addressing mode activated at the time of contingency definition (AMODE) is automatically activated by the operating system at the time the contingency routine is executed.

The **ENACO** macro can be invoked both in a basic process and in a contingency process. The basic process is always the associated independent process initiated first (with SET-LOGON-PARAMETERS).

As soon as a routine is no longer to be used as a contingency process, the user may disable the contingency definition by means of the **DISCO** macro, specifying the name of the relevant contingency process. This causes the contingency definition to be deleted.

Any subsequent **POSSIG** and **SOLSIG** macros referring to a deleted contingency process are rejected. Any **POSSIG** or **SOLSIG** macros whose entries are still in the relevant event item queue when the contingency definition is deleted, are not affected. Consequently, the contingency process will still be activated when the associated event occurs.

Unlike the scope of an event item, the scope of a contingency process is always local; its application is therefore restricted to the defining task. A task can use up to 400 contingency processes simultaneously.

An ID for the name of the contingency process is provided by the system to the defining task at an address. The use of this ID speeds up processing; it is mandatory in the **POSSIG** and **SOLSIG** macros. If a contingency process is deleted and subsequently specified again under the same name, the ID may be different from that supplied on the previous occasion.

Start and termination of a contingency process

The user enables the start of a contingency process by specifying its ID as an operand in a **POSSIG** or **SOLSIG** macro (see page 115). The link with the activating event is thus established.

A contingency process is activated and, subject to its priority, starts its execution under the following conditions:

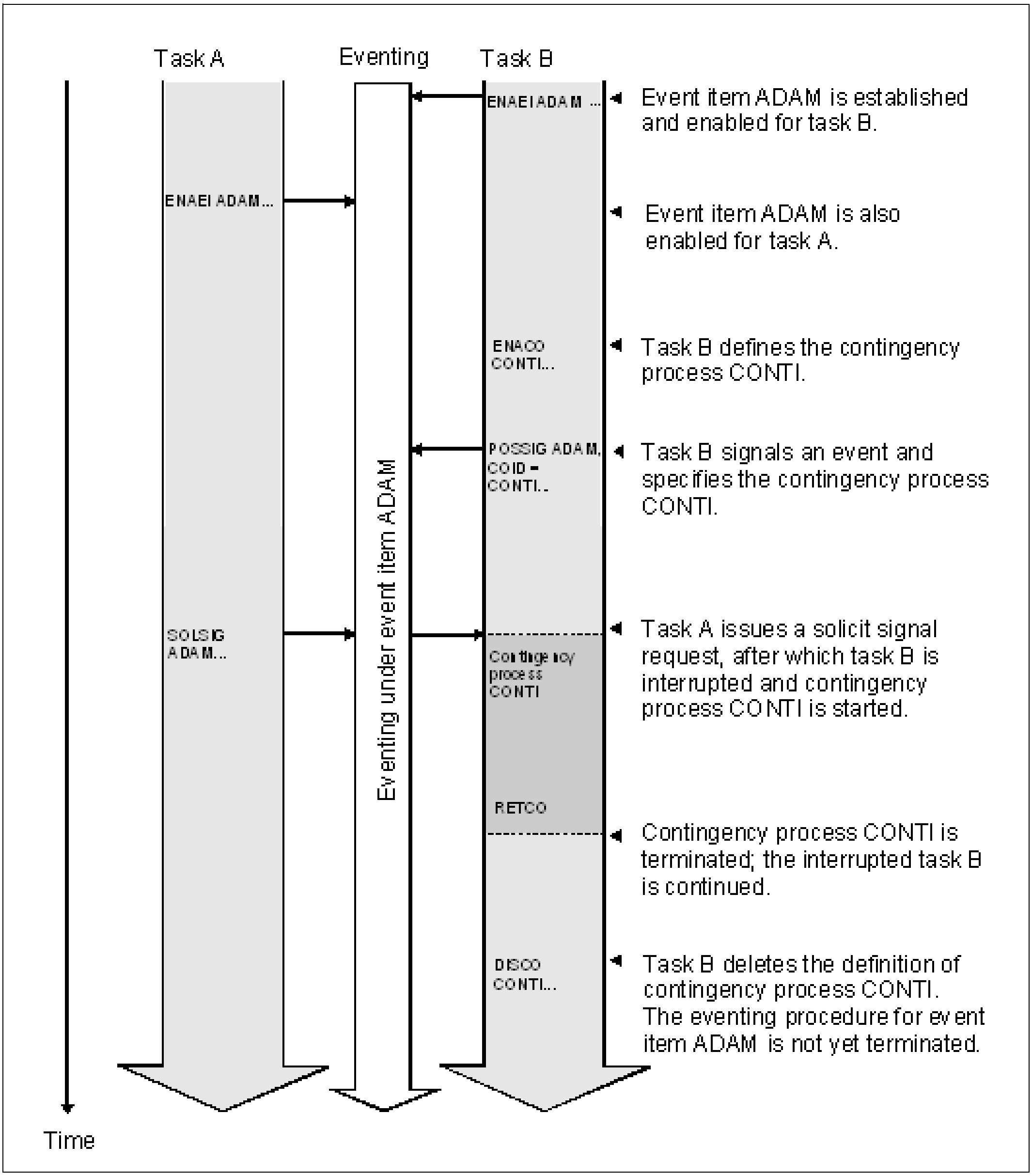
- Asynchronous eventing
  A contingency process was specified in the **SOLSIG** macro to make a solicit signal request and the signal has been received (figure 17 on page 101) or the time interval has elapsed.
  The contingency process is also started if the basic process has deleted the event item (**DISEI**).
- Acknowledgment of a signal
  A contingency process was specified in the **POSSIG** macro to make a post signal request and the signal was solicited (figure 19 on page 113) or the time interval has elapsed. The contingency process is also started if the basic process has previously deleted the event item (**DISEI**).

The system starts a contingency process under the following conditions:

- No process with a higher priority is activated or started.
- No process with equal priority precedes it in the queue.

At its start, the contingency process must first define a base register.

The contingency process is started with the access level valid during the associated **SOLSIG** or **POSSIG** call. The contingency routine is started by the operating system with the same addressing mode which was active at the time of contingency definition (**ENACO** or **STXIT**). Upon termination of the contingency process, the interrupted task is reassigned the access level it had before the interrupt.



Event item ADAM is established and enabled for task B.

Event item ADAM is also enabled for task A.

Task B defines the contingency process CONTI.

Task B signals an event and specifies the contingency process CONTI.

Task A issues a solicit signal request, after which task B is interrupted and contingency process CONTI is started.

Contingency process CONTI is terminated; the interrupted task B is continued.

Task B deletes the definition of contingency process CONTI. The eventing procedure for event item ADAM is not yet terminated.

Figure 19: Flow diagram of a contingency process
Here it is used to return an acknowledgment.

The execution of a contingency process is terminated when the process issues the **RETCO** macro. Depending on priority, control will then be returned to the task (basic process) or to another contingency process.

Contingency process priorities

If several contingency processes have been defined, the use of priority levels, assigned with the contingency definition, and the modification of priority levels during execution provide for coordination facilities when several events occur simultaneously.

Permissible range of priority levels:

Basic process	priority 0-127	default value: 0
Contingency process	priority 1-127	default value: 1

By default, a contingency process has a higher priority level than the basic process. The basic process is interrupted when the contingency process is initiated, and is not continued until the contingency process is terminated. A current contingency process can also be interrupted by another contingency process having a higher priority level (figure 20 on page 116). If several contingency processes have the same priority level, they are entered in a queue and processed, by default, on the FIFO principle (first in, first out).

Basic processes and contingency processes can change their priority during processing by means of the **LEVCO** macro. At the same time a choice can be made between the LIFO and FIFO method. A process with a new priority is placed among tasks of the same priority according to the selected queueing method.

– FIFO queue processing (first in, first out):
  After process activation or after execution of the **LEVCO** macro, the process is placed at the end of the queue for processes having the same priority level. Any processes preceding it in the queue will be started before it. If a process lowers its priority during processing using the FIFO method, the process may possibly be interrupted by processes having the same priority. This can be prevented by specifying the LIFO method in the **LEVCO** macro.

– LIFO queue processing (last in, first out):
  After process activation or after execution of the **LEVCO** macro, the process is placed at the beginning of the queue for processes having the same priority. Thus, only processes with a higher priority level can be executed before this process.

For further details on the interruptibility of contingency processes see the relevant note in the **CONTXT** macro description.

It is the user's responsibility to assign priorities so as to provide for correct nesting of processes. When the priority level of the basic process is increased special care must be taken to prevent any subsequent contingency processes from being blocked until the basic process terminates. If, in such a situation (the active process has a higher priority than contingency processes already waiting in the queue), the active process is placed in the wait state with the **SUSPEND** macro, this does *not* cause the waiting contingency processes to start.

Restrictions

> A process is not allowed to lower its priority to a level where it is below the priority level of a process already started (and interrupted). A previously started and interrupted process is only allowed to restart after the interrupting contingency process is terminated.

Access to interrupted processes

Every contingency process has its own set of registers.

A contingency process has access to the registers of the process it has interrupted or to those of the basic process. The **CONTXT** macro enables the contingency process to read or modify the floating-point registers, the program counter and the general registers of the interrupted contingency process or of the basic process.

Information transfer to contingency processes

A contingency process, when activated, may obtain up to three types of information:

Register R1 — may contain a **contingency message** (4 bytes). It can be specified in the **ENACO** macro or - at a later stage, by overwriting - in the **POSSIG** or **SOLSIG** macro (figure 18 on page 102). Its format and meaning can be defined by the user without any restriction.

Register R2 — always contains the **event information code** (2 bytes), which specifies the conditions that caused the contingency process to be initiated. Its form and meaning are predefined (see table 8 on page 117).

Register R3 (+ R4) — may contain a **post code** (4 or 8 bytes). If the post code is 8 bytes (2 words) long, the second word is entered in register R4. The post code can be specified in the **POSSIG** macro (figure 18 on page 102). Its format and meaning can be defined by the user subject to existing conventions (see page 96).

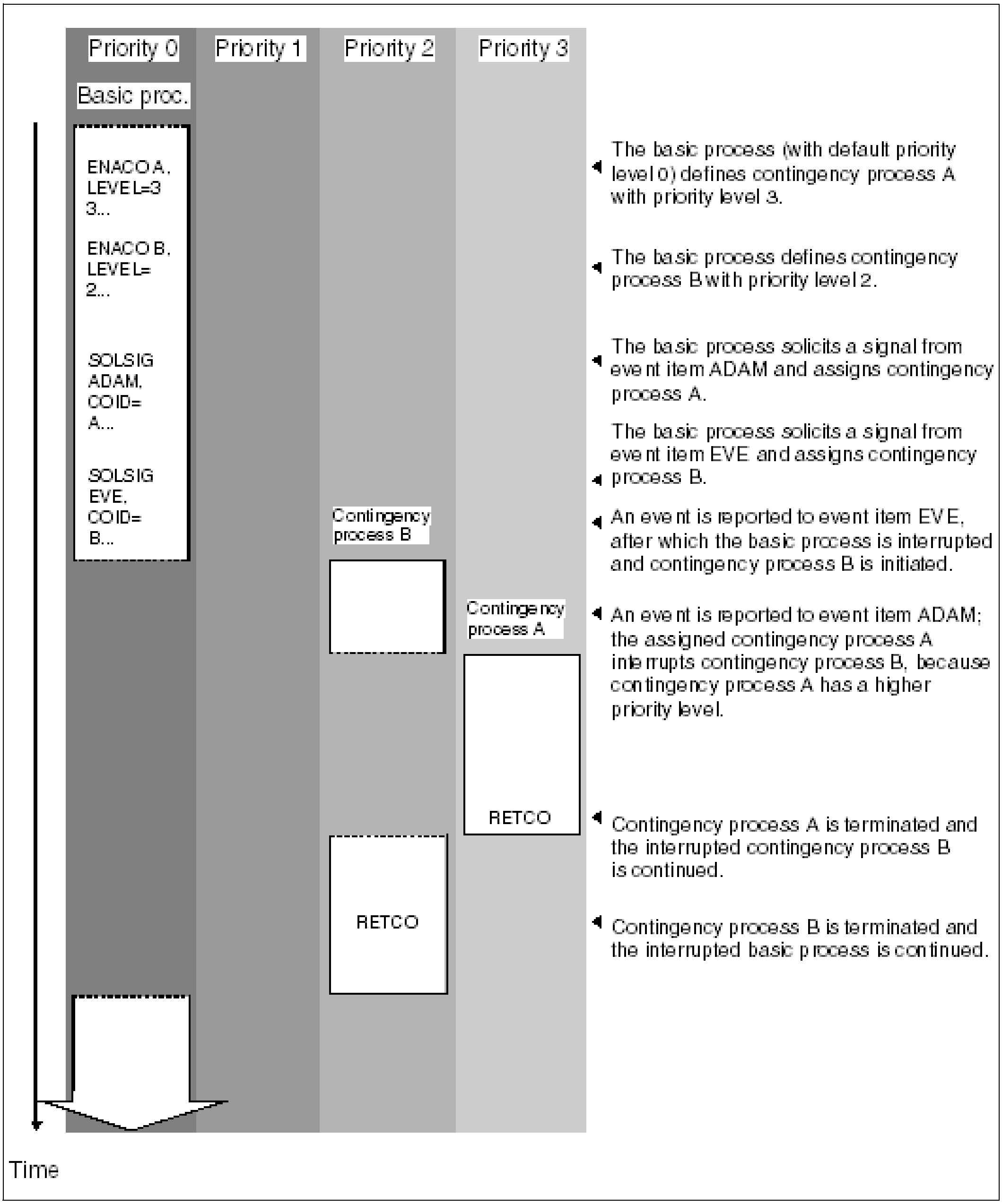

Priority 0
Priority 1
Priority 2
Priority 3
Basic proc.
ENACO A, LEVEL=3 3...
ENACO B, LEVEL= 2...
SOLSIG ADAM, COID= A...
SOLSIG EVE, COID= B...
Contingency process B
Contingency process A
RETCO
RETCO
Time
The basic process (with default priority level 0) defines contingency process A with priority level 3.
The basic process defines contingency process B with priority level 2.
The basic process solicits a signal from event item ADAM and assigns contingency process A.
The basic process solicits a signal from event item EVE and assigns contingency process B.
An event is reported to event item EVE, after which the basic process is interrupted and contingency process B is initiated.
An event is reported to event item ADAM; the assigned contingency process A interrupts contingency process B, because contingency process A has a higher priority level.
Contingency process A is terminated and the interrupted contingency process B is continued.
Contingency process B is terminated and the interrupted basic process is continued.

Figure 20: Prioritized execution of contingency processes

The **event information code** in register R2 consists of the event switch ES in the rightmost byte and the information indicator II in the leftmost byte.

II	ES	Meaning
X'00'	X'00'	Event has occurred as expected. Neither post code nor contingency message is available.
X'04'	X'00'	Event has occurred as expected. Contingency message is available; post code is not available.
X'08'	X'00'	Event has occurred as expected. Contingency message is not available. Post code (length = 4 bytes) has been entered in register R3.
X'0C'	X'00'	Event has occurred as expected. Both contingency message and post code (length = 4 bytes) are available.
X'28'	X'00'	Event has occurred as expected. Contingency message is not available. Post code (length = 8 bytes) has been entered in registers R3 and R4.
X'2C'	X'00'	Event has occurred as expected. Both contingency message and post code (length = 8 bytes) are available.
X'00'	X'04'	Event did not occur within the specified period of time. Neither contingency message nor post code is available.
X'04'	X'04'	Event did not occur within the specified period of time. Contingency message is available; post code is not available.
X'08'	X'04'	Event did not occur within the specified period of time. Contingency message is not available; post code (4 bytes) has been entered in register R3.
X'0C'	X'04'	Event did not occur within the specified period of time. Both contingency message and post code (4 bytes) are available.
X'10'	X'04'	The event item was disabled (DISEI) before the event occurred. Neither contingency message nor post code is available.
X'14'	X'04'	The event item was disabled (DISEI) before the event occurred. Contingency message is available; post code is not available.
X'18'	X'04'	The event item was disabled (DISEI) before the event occurred. Contingency message is not available; post code (4 bytes) has been entered in register R3.
X'1C'	X'04'	The event item was disabled (DISEI) before the event occurred. Both contingency message and post code (4 bytes) are available.
X'28'	X'04'	Event did not occur within the specified period of time. Contingency message is not available; post code (8 bytes) has been entered in registers R3 and R4.
X'2C'	X'04'	Event did not occur within the specified period of time. Both contingency message and post code (8 bytes) are available.
X'38'	X'04'	The event item was disabled (DISEI) before the event occurred. Contingency mes-sage is not available; post code (8 bytes) has been entered in registers R3 and R4.

Table 8: Event information codes

II	ES	Meaning
X'3C'	X'04'	The event item was disabled (DISEI) before the event occurred. Both contingency message and post code (8 bytes) are available.

Table 8: Event information codes

Example: Asynchronous operation

Part 1: Interactive job with PCOSOL1 program

The PCOSOL1 program defines two event items (ADAM and EVE) and two contingency processes (CONTA and CONTE). The program solicits a signal from each event item by means of the **SOLSIG** macro, and specifies a contingency process each time (asynchronous operation). In part 1 of the example, no **POSSIG** calls are issued to the event items. Contingency process CONTA is started because its waiting time has elapsed (60 seconds). Contingency process CONTE is started because the program deletes event item EVE.

Program PCOSOL1

```
PCOSOL1  START
         PRINT NOGEN
         BALR  5,0
         USING *,5
         ENAEI EINAME=ADAM,SCOPE=GROUP,EIIDRET=ABBRADAM  ———————————————  (1)
         ENAEI EINAME=EVE,SCOPE=GROUP,EIIDRET=ABBREVE
         ENACO CONAME=A,COADAD=CONTAAD,COIDRET=ABBRA  ——————————————————  (2)
         ENACO CONAME=E,COADAD=CONTEAD,COIDRET=ABBRE
         GDATE TOD=TIMEBAS1
         SOLSIG EIID=ABBRADAM,COID=ABBRA,LIFETIM=60  ———————————————————  (3)
         CL    15,NULL
         BNE   ERROR
         SOLSIG EIID=ABBREVE,COID=ABBRE  ———————————————————————————————  (4)
         CL    15,NULL
         BNE   ERROR
         WROUT MLDBAS1,ERROR
         VPASS 120  ————————————————————————————————————————————————————  (5)
CONNECT  GDATE TOD=TIMEBAS2
         CHKEI EIID=ABBRADAM
         ST    1,WSADAM
         ST    15,WSARC
         CHKEI EIID=ABBREVE
         ST    1,WSEVE
         ST    15,WSERC
         WROUT MLDBAS2,ERROR
         DISCO COID=ABBRA  —————————————————————————————————————————————  (6)
         DISCO COID=ABBRE
         GDATE TOD=TIMEBAS3
         DISEI EIID=ABBRADAM ------------------------------------------  (7)
         DISEI EIID=ABBREVE
         WROUT MLDBAS3,ERROR
DTH1     TERM
*
CONTA    BALR  6,0 ----------------------------------------------------  (8)
         USING *,6
         GDATE TOD=TIMEA
         ST    2,INFOADAM
         WROUT MLDCONA,ERROR
         CONTXT SAVE=LASTREG,PROCESS=LAST
         ST    15,RCCONTXT
         RETCO

*
         DS    0F
LASTREG  DS    CL68
INFOADAM DS    F

MLDCONA  DC    Y(ENDCA-MLDCONA)
         DS    L2
         DC    X'01'
         DC    'TIMEOUT CONTINGENCY A AT '
TIMEA    DS    CL8
ENDCA    EQU   *
*
CONTE    BALR  7,0 ----------------------------------------------------  (9)
         USING *,7
         GDATE TOD=TIMEE
         WROUT MLDCONE,ERROR
         ST    2,INFOEVE
         RETCO
*
INFOEVE  DS    F
MLDCONE  DC    Y(ENDCE-MLDCONE)
         DS    L2
         DC    X'01'
         DC    'TIMEOUT CONTINGENCY E AT '
TIME    DS    CL8
ENDCE    EQU   *
*
ERROR    CDUMP2 SCOPE=AREA
         TERM

CONTAAD  DC    A(CONTA)
CONTEAD  DC    A(CONTE)
NULL     DC    F'0'
```

```
RCCONTXT DS    F
*
ABBRADAM DS    F
ABBREVE  DS    F
ABBRA    DS    F
ABBRE    DS    F
*
MLDBAS1  DC    Y(ENDBAS1-MLDBAS1)
         DS    L2
         DC    X'01'
         DC    'BOTH SOLSIGS ISSUED AT '
TIMEBAS1 DS    CL8
ENDBAS1  EQU   *
MLDBAS2  DC    Y(ENDBAS2-MLDBAS2)
         DC    X'000001'
         DC    'QUEUES CHECKED AT '
TIMEBAS2 DS    CL8
ENDBAS2  EQU   *
MLDBAS3  DC    Y(ENDBAS3-MLDBAS3)
         DC    X'000001'
         DC    'EVENT ITEMS DISABLED AT '
TIMEBAS3 DS    CL8
ENDBAS3  EQU   *
*
WSADAM   DS    F
WSARC    DS    F
WSEVE    DS    F
WSERC    DS    F
ENDE     EQU   *
         END
```

Basic process

(1) Event items ADAM and EVE are defined. The addresses of the IDs are ABBRADAM and ABBREVE.

(2) The CONTA routine is defined as contingency process A. The start address (CONTA) is stored at the address CONTAAD; the address of the ID is ABBRA. The same applies to the following definition of the CONTE routine as contingency process E.

(3) The program solicits a signal from event item ADAM via a **SOLSIG** call and specifies contingency process CONTA. If the signal has not arrived within a waiting period of 60 seconds, event item ADAM is to start contingency process CONTA. Program execution is continued after this **SOLSIG** call.

(4) With another **SOLSIG** call, a signal is solicited from EVE. For contingency process CONTE, the default waiting time (10 minutes) applies.

(5) For demonstration purposes, the program is made to wait by issuing the **VPASS** macro. It is then interrupted by contingency process CONTA, whose waiting period has elapsed. After CONTA has terminated, the program goes on to check the event queues of ADAM and EVE.

(6) The program deletes the definitions of contingency processes CONTA and CONTE. At this point, the **SOLSIG** call to EVE with contingency process CONTE still resides in the event queue of EVE. CONTE can still be started, but another **SOLSIG** call in which CONTE is specified would be rejected at this point.

(7) The program first deletes event item ADAM, and then EVE. As soon as EVE is deleted, the system starts contingency process CONTE.

Contingency process CONTA

(8) As a contingency process has its own register set, a base register must be defined and loaded at the start. Registers R1, R2 and R3 are not suitable, since they might be used by the system during eventing (see “Information transfer to contingency processes” on page 115). The event information code that was transferred to the contingency process in register 2 is now stored under the address INFOADAM. For demonstration purposes, the registers of the interrupted process (in this case the basic process) are transferred to the LASTREG area by means of the **CONTXT** macro. The contingency process terminates with the **RETCO** call.

Contingency process CONTE

(9) A base register is defined and loaded. The event information code is stored under INFOEVE, and contingency process CONTE is terminated with **RETCO**.

Runtime log of the interactive job with PCOSOL1

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,pcosol1), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,pcosol1)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 1238 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 171 MSEC
//end
%  ASS6012 END OF ASSEMBH
```

```
/load-executable-program library=macexmp.lib,element-or-symbol=pcosol1, -
/     test-options=*aid
%  BLS0523 ELEMENT 'PCOSOL1', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'PCOSOL1', VERSION ' ' OF '<date> <time>' LOADED
/%in dth1;%r
BOTH SOLSIGS ISSUED AT 16:06:33 ──────────────────────────────────────── (10)
TIMEOUT CONTINGENCY A AT 16:07:33 ────────────────────────────────────── (11)
QUEUES CHECKED AT 16:07:33 ───────────────────────────────────────────── (12)
TIMEOUT CONTINGENCY E AT 16:07:33 ────────────────────────────────────── (13)
EVENT ITEMS DISABLED AT 16:07:33
STOPPED AT LABEL: DTH1 , SRC_REF: 380, SOURCE: PCOSOL1 , PROC: PCOSOL1
/%d infoadam %x, %@(wsadam) -> %xl8 ──────────────────────────────────── (14)
*** TID: 00340180 *** TSN: 1PKB *****************************************
CURRENT PC: 00000162    CSECT: PCOSOL1  *********************************
V'00000204' = INFOADAM + #'00000000'
00000204 (00000000) 00000004                              ....
V'00000334' = PCOSOL1  + #'00000334'
00000334 (00000334) 000000D0 30000000                     ........
/%d infoeve %x, %@(wseve) -> %xl8 ────────────────────────────────────── (15)
V'0000025C' = INFOEVE  + #'00000000'
0000025C (00000000) 10000004                              ....
V'0000033C' = PCOSOL1  + #'0000033C'
0000033C (0000033C) 00000001 28000000                     ........
/%r
```

(10) From each of the two event items the program solicits a signal.

(11) The waiting period for contingency process CONTA has elapsed.

(12) The status of the POSSIG queue and the SOLSIG queue of each event item is checked.

(13) Contingency process CONTE is executed because event item EVE has been disabled by the basic process.

(14) Event item ADAM:
The event information code X'04' means that the waiting period has elapsed without the event having occurred.
The return information X'30' of the **CHKEI** macro (in the `WSARC` field) means that the event queues are empty: the SOLSIG entry was therefore removed, because contingency process CONTA was completed.

(15) Event item EVE:
The event information code X'10000004' means that the event item was disabled before an event occurred.

The return code X'28' of the **CHKEI** macro (in the `WSERC` field) means that the SOLSIG queue is not empty. Field WSEVE: X'01' specifies the number of existing entries, in this case one entry: the SOLSIG entry still exists, because the contingency process CONTE has not yet been started.

Part 2: Interactive job using the program PCOSOL2
ENTER job ENTER.POSA using the program POSA
ENTER job ENTER.POSE using the program POSE

PCOSOL2 program

The PCOSOL2 program differs from PCOSOL1 only in that it starts two ENTER jobs after issuing the two **SOLSIG** calls to event items ADAM and EVE. Only that part of the source program PCOSOL2 program which contains the change is listed below; the rest is identical with the PCOSOL1 program.

```
PCOSOL2  START
         PRINT NOGEN
         BALR  5,0
         USING *,5
         ENAEI EINAME=ADAM,SCOPE=GROUP,EIIDRET=ABBRADAM
         ENAEI EINAME=EVE,SCOPE=GROUP,EIIDRET=ABBREVE
         ENACO CONAME=A,COADAD=CONTAAD,COIDRET=ABBRA
         ENACO CONAME=E,COADAD=CONTEAD,COIDRET=ABBRE
         GDATE TOD=TIMEBAS1
         SOLSIG EIID=ABBRADAM,COID=ABBRA,LIFETIM=60
         CL    15,NULL
         BNE   ERROR
         SOLSIG EIID=ABBREVE,COID=ABBRE
         CL    15,NULL
         BNE   ERROR
         WROUT MLDBAS1,ERROR
         ENTER 'ENTER.POSE,JOB-CLASS=JCB00050'
         ENTER 'ENTER.POSA,JOB-CLASS=JCB00050'
         VPASS 90
CONNECT  GDATE TOD=TIMEBAS2
         ...
```

POSA program and ENTER file ENTER.POSA

The ENTER job ENTER.POSA starts the POSA program. The POSA program enables event item ADAM and sends a **POSSIG** call to ADAM. Before deleting the event item, it waits 250 seconds by issuing the **VPASS** macro in order to prevent the entry from being removed from the event queue before the PCOSOL2 program can call the **SOLSIG** macro.

```
POSA     START
         PRINT NOGEN
         BALR  5,0
         USING *,5
         ENAEI EINAME=ADAM,SCOPE=GROUP,EIIDRET=ABBRADAM
         GDATE TOD=TIMEA
         POSSIG EIID=ABBRADAM
         CL    15,=F'0'
         BE    OK
ERROR    EQU   *
*****  Error handling  *****
         TERM
*

OK       WROUT MLDPOSA,ERROR
         VPASS 250
         DISEI EIID=ABBRADAM
         TERM
ABBRADAM DS    F
MLDPOSA  DC    Y(ENDE-MLDPOSA)
         DC    X'000001'
         DC    'POSSIG FOR ADAM ISSUED AT '
TIMEA    DS    CL8
ENDE     EQU   *
         END
```

ENTER file ENTER.POSA

```
/.POSA SET-LOGON-PARAMETERS
/ASSIGN-SYSDTA *SYSCMD
/START-ASSEMBH
//COMPILE SOURCE=*LIBRARY-ELEMENT(MACEXMP.LIB,POSA), -
//        COMPILER-ACTION=MODULE-GENERATION(MODULE-FORMAT=LLM), -
//        MODULE-LIBRARY=MACEXMP.LIB, -
//        LISTING=PARAMETERS(OUTPUT=*LIBRARY-ELEMENT(MACEXMP.LIB,POSA))
//END
/ASSIGN-SYSDTA *PRIMARY
/ASSIGN-SYSOUT PROT.POSA
/START-EXECUTABLE-PROGRAM LIBRARY=MACEXMP.LIB,ELEMENT-OR-SYMBOL=POSA
/EXIT-JOB
```

POSE program and ENTER file ENTER.POSE

The ENTER job ENTER.POSE starts the POSE program. The program enables event item EVE and issues a **POSSIG** call to EVE. For the same reason specified for POSA, it waits before issuing the **DISEI** call.

```
POSE     START
         PRINT NOGEN
         BALR  5,0
         USING *,5
         ENAEI EINAME=EVE,SCOPE=GROUP,EIIDRET=ABBREVE
         GDATE TOD=TIMEE
         POSSIG EIID=ABBREVE
         CL    15,=F'0'
         BE    OK
ERROR    EQU   *
*****  Error handling  *****
         TERM
*
OK       WROUT MLDPOSE,ERROR
         VPASS 150
         DISEI EIID=ABBREVE
         TERM
ABBREVE  DS    F
MLDPOSE  DC    Y(END-MLDPOSE)
         DC    X'000001'
         DC    'POSSIG FOR EVE ISSUED AT '
TIMEE    DS    CL8
ENDE     EQU   *
         END
```

ENTER file ENTER.POSE as for ENTER.POSA

Runtime log of the interactive job PCOSOL2 and the two ENTER jobs

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,pcosol2), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,pcosol2)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 1217 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 166 MSEC//END
%  ASS6012 END OF ASSEMBH
```

```
/load-executable-program library=macexmp.lib,element-or-symbol=pcosol2, -
/     test-options=*aid
%  BLS0523 ELEMENT ’PCOSOL1’, VERSION ’@’ FROM LIBRARY
   ’:2OSG:$QM212.MACEXMP.LIB’ IN PROCESS
%  BLS0524 LLM ’PCOSOL1’, VERSION ’ ’ OF ’<date> <time>’ LOADED
/%in dth1;%r
BOTH SOLSIGS ISSUED AT 17:09:06 ---------------------------------------  (1)
%  JMS0066 JOB ’POSE’ ACCEPTED ON 12-01-20 AT 17:09, TSN = 1PY0
%  JMS0066 JOB ’POSA’ ACCEPTED ON 12-01-20 AT 17:09, TSN = 1PY1
```

In the meantime, the ENTER jobs POSE and POSA output the following messages to their output log:

ENTER.POSE:

```
POSSIG FOR EVE ISSUED AT 17:09:21 --------------------------------------  (2)
```

ENTER.POSA:

```
POSSIG FOR ADAM ISSUED AT 17:09:22 -------------------------------------  (3)
```

The interactive job continues as follows

```
TIMEOUT CONTINGENCY A AT 17:10:36 -------------------------------------  (4)
TIMEOUT CONTINGENCY E AT 17:10:36
QUEUES CHECKED AT 17:10:36
EVENT ITEMS DISABLED AT 17:10:36
STOPPED AT LABEL: DTH1 , SRC_REF: 414, SOURCE: PCOSOL2 , PROC: PCOSOL2
/%d infoadam %x, %@(wsadam) -> %x18 -----------------------------------  (5)
*** TID: 00340180 *** TSN: 1PKB *********************************************
CURRENT PC: 000001D2    CSECT: PCOSOL2  *************************************
V’00000274’ = INFOADAM + #’00000000’
00000274 (00000000) 00000000                               ....
V’000003A4’ = PCOSOL2  + #’000003A4’
000003A4 (000003A4) 00000140 30000000                      ... ....
/%d infoeve %x, %@(wseve) -> %x18
V’000002CC’ = INFOEVE  + #’00000000’
000002CC (00000000) 00000000                               ....
V’000003AC’ = PCOSOL2  + #’000003AC’
000003AC (000003AC) 00000158 30000000                      ........
/%r
```

(1) The PCOSOL2 program defines the two event items ADAM and EVE and addresses a **SOLSIG** call to each of them. Subsequently, it starts ENTER job ENTER.POSE and then ENTER job ENTER.POSA.

(2) ENTER job ENTER.POSE starts the program POSE, which enables event item EVE and issues a **POSSIG** call.

(3) ENTER job ENTER.POSA starts the program POSA, which enables event item ADAM and issues a **POSSIG** call.

(4) Contingency processes CONTA and CONTE both have priority 1. The system starts with the contingency process whose corresponding event item is the first to receive a **SOLSIG** and a **POSSIG** call. In this case, a **SOLSIG** call already exists for each event item, and a **POSSIG** call arrives earlier for the EVE event item than for ADAM. Therefore, contingency process CONTE is started first, followed by contingency process CONTA.

(5) On completion of the two contingency processes, the program of the basic process is continued. It checks the queues of both event items: they are empty.
(After the **CHKEI** macro, the fields WSERC and WSARC contain the value X'30000000'.) The event information codes of both contingency processes are zero: the expected event has occurred.

Part 3: Interactive job using the PCOSOL3 program
ENTER jobs ENTER.POSA and ENTER.POSE as in part 2

PCOSOL3 program

The PCOSOL3 program differs from PCOSOL1 and PCOSOL2 in that it starts the two ENTER jobs at the beginning, before issuing the two **SOLSIG** calls. Only that part of the source program PCOSOL3 containing the change is shown here; the rest is identical with the PCOSOL1 program.

```
PCOSOL3  START
         PRINT NOGEN
         BALR  4,0
         USING *,4
         ENTER 'ENTER.POSE,JOB-CLASS=JCB00050'
         ENTER 'ENTER.POSA,JOB-CLASS=JCB00050'
         ENAEI EINAME=ADAM,SCOPE=GROUP,EIIDRET=ABBRADAM
         ENAEI EINAME=EVE,SCOPE=GROUP,EIIDRET=ABBREVE
         ENACO CONAME=A,COADAD=CONTAAD,COIDRET=ABBRA
         ENACO CONAME=E,COADAD=CONTEAD,COIDRET=ABBRE
         VPASS 40
         GDATE TOD=TIMEBAS1
         SOLSIG EIID=ABBRADAM,COID=ABBRA,LIFETIM=60
         CL    15,NULL
         BNE   ERROR
         SOLSIG EIID=ABBREVE,COID=ABBRE
         CL    15,NULL
         BNE   ERROR
         WROUT MLDBAS1,ERROR
         VPASS 90
CONNECT  GDATE TOD=TIMEBAS2
         ...
```

ENTER job ENTER.POSA using the POSA program: see part 2
ENTER job ENTER.POSE using the POSE program: see part 2

Runtime log of the interactive job with PCOSOL3 and the two ENTER jobs

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,pcosol3), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,pcosol3)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 1260 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 168 MSEC
//end
%  ASS6012 END OF ASSEMBH
/load-executable-program library=macexmp.lib,element-or-symbol=pcosol3, -
/     test-options=*aid
%  BLS0523 ELEMENT 'PCOSOL3', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'PCOSOL3', VERSION ' ' OF '<date> <time>' LOADED
/%in dth1;%r
%  JMS0066 JOB 'POSE' ACCEPTED ON 12-01.20 AT 17:20, TSN = 1PY7 ———————  (1)
%  JMS0066 JOB 'POSA' ACCEPTED ON 12-01-20 AT 17:20, TSN = 1PY8
```

In the meantime, the ENTER jobs POSE and POSA output the following messages to their output log:

ENTER.POSE:

```
POSSIG FOR EVE ISSUED AT 17:20:19 ————————————————————————————————————  (2)
```

ENTER.POSA:

```
POSSIG FOR ADAM ISSUED AT 17:20:20 ———————————————————————————————————  (3)
```

The interactive job continues as follows

```
TIMEOUT CONTINGENCY A AT 17:20:52 ————————————————————————————————————  (4)
TIMEOUT CONTINGENCY E AT 17:20:52
BOTH SOLSIGS ISSUED AT 17:20:52 ——————————————————————————————————————  (5)
QUEUES CHECKED AT 17:22:22
EVENT ITEMS DISABLED AT 17:22:22
STOPPED AT LABEL: DTH1 , SRC_REF: 417, SOURCE: PCOSOL3 , PROC: PCOSOL3
/%d infoadam %x, %@(wsadam) -> %xl8
*** TID: 00340180 *** TSN: 1PKB *********************************************
CURRENT PC: 000001DA    CSECT: PCOSOL3  *************************************
V'0000027C' = INFOADAM + #'00000000'
0000027C (00000000) 00000000                                      ....
V'000003AC' = PCOSOL3  + #'000003AC'
```

```
000003AC (000003AC) 00000148 30000000                      ........
/%d infoeve %x, %@(wseve) -> %x18
V'000002D4' = INFOEVE  + #'00000000'
000002D4 (00000000) 00000000                               ....
V'000003B4' = PCOSOL3  + #'000003B4'
000003B4 (000003B4) 00000160 30000000                      ...¯....
/%r
```

(1) The PCOSOL3 program first starts ENTER job ENTER.POSE (TSN 02C1) and then ENTER job ENTER.POSA (TSN 02C2).
(Event items ADAM and EVE are, for demonstration purposes, enabled by the program after some time has elapsed; the **SOLSIG** calls for ADAM and EVE are issued later.)

(2) The PCOPOSE program (started in ENTER job ENTER.POSE) defines event item EVE and issues a **POSSIG** call for EVE.

(3) The PCOPOSA program (started in ENTER job ENTER.POSA) defines event item ADAM and issues a **POSSIG** call for ADAM.

(4) As soon as the PCOSOL3 program (in the interactive job) has issued the **SOLSIG** call for event item ADAM, it is interrupted by the start of contingency process A, since a **POSSIG** call already exists for ADAM. On completion of A, the program may continue and issue the **SOLSIG** call for EVE; a **POSSIG** call also exists here already, and the program is interrupted again until contingency process E is terminated. Only then can it output the message concerning the **SOLSIG** calls.

(5) The program checks the queues of both event items: they are empty. (After the **CHKEI** macro, the fields WSERC and WSARC contain the value X'30000000'.) The event information codes of both contingency processes are zero: the expected event has occurred.

For a further **example** see the **POSSIG** macro description on page 736.

4.3.7 STXIT procedure with contingency processing

Macro	Brief description
CONTXT	Reads or writes in the registers of the process it interrupted or of the basic process etc.
EXIT	Terminates an STXIT process
LEVCO	Changes the processing level (priority) of the calling STXIT process during execution
SETIC	Sets or resets the interval timer (CPU time/real time)
STXIT	Specifies user-written STXIT processes, with which the system continues processing when a program interrupt occurs

General

The following events usually influence the continuation of the program run:

- invalid SVC, invalid operation code,
- data error, overflow,
- end of the program runtime, break/escape interrupts, INFORM-PROGRAM command,
- address errors.

The STXIT procedure enables users to react to such interrupts with their own (STXIT) routines: if a program interrupt should occur, the STXIT routine is executed and then (in normal circumstances) the interrupted basic process resumed. A basic process is a currently executing program that was started with the START-PROGRAM command. The STXIT routines are components of the program, but are executed as separate tasks.

Examples

- A program error (invalid SVC, invalid operation code, data error, overflow,...) usually causes abnormal program termination. This can be avoided with an STXIT routine.
- A break (K2 key) usually causes the program to be interrupted (the operating system now expects BS2000 commands). The programmer can use the STXIT routine to anticipate an interrupt in the program.

Description of the STXIT procedure

- STXIT management

  The interrupt events to be processed with the STXIT procedure are put into STXIT event classes (see table at the end of this section). The user can assign an STXIT routine to every event class by means of the **STXIT** macro. The specified assignments are entered in the system in an STXIT management block. In further **STXIT** calls, the user can modify or extend the entered assignments in the created STXIT management block, or create a new STXIT management block with further assignments. The STXIT management blocks are chained together.

  Using the STXIT procedure, up to 100 STXIT management blocks can be created in one program (or program system). This enables several STXIT routines to coexist for the same event class. This extension has advantages for applications in program systems because, when an event occurs, each subprogram can be supplied with the control needed for this particular event.

- STXIT routine

  An STXIT routine is a control section in a program. It contains the programmer's reaction to the interrupt event. The routine is terminated with the **EXIT** macro. The STXIT routine starts its execution when the interrupt event occurs.
  The actual interrupt event is described with the aid of an event code and communicated to the program via register 3 (see table 9 on page 137).

  When the interrupt event occurs, the STXIT routine executes in its own process (STXIT process). In this case, the operating system uses the registers R1, R3 and R4 to communicate with the routine.
  The register contains the following information:

R1: STXIT message, where STXMSG is specified for STXIT operand.

R3: in the rightmost byte: event code of the interrupt event (see table 9 on page 137)

R4: in the rightmost byte: SVC number for SVC event class or function-key code for ESCPBRK event class. Function-key codes can be taken from the “TIAM” manual [16] or the table “Standardized function key codes” on page 1153. This functionality is only provided if the TIAM partner is a terminal and not an application (e.g. OMNIS).

- The STXIT process

– Definition of an STXIT process

STXIT processes are contingency processes (i.e. they have their own process control block (PCB) and therefore their own set of registers, and their own processing level). The start address of the STXIT process is the address of the STXIT routine that was specified in the **STXIT** macro. By default, the STXIT process is assigned the highest processing level (127).

– Execution of STXIT processes

An STXIT process is normally started as soon as an interrupt event occurs. If further interrupt events occur during the run, a process queue is created. The processing level and queueing method (LIFO/FIFO) for an STXIT process determine its place in the process queue.

LIFO (Last In First Out) means that:

After a process has been activated, it is entered in the queue at the head of processes with the same processing level (priority). Only processes with a higher processing level can precede it.
STXIT processes for event classes with time-independent events (such as program check, unrecoverable program errors,...., program termination) are entered according to the LIFO principle (see table 9 on page 137).

FIFO (First In First Out) means that:

After a process has been activated it is entered in the queue as the last of the processes with the same priority. Processes that were activated at an earlier time and have the same processing level as this process, as well as processes with a higher processing level, may precede it in the queue.
STXIT processes for the event classes with time-dependent events (timer for real time, timer for CPU time, end of program runtime) are entered according to the FIFO principle (see table 9 on page 137).

The interaction of several STXIT processes is explained below, using the current basic process for demonstration purposes. For simplicity's sake it is assumed that all STXIT processes have the same processing level, and that the basic process is continued after the STXIT processes are terminated.

In the example it is assumed either that:

a) only one STXIT routine is ever assigned to one event class in the program or program system, or that

b) several STXIT routines are assigned to the event classes in the program or program system (STXIT parallelism).

For a):

When an interrupt event occurs, the basic process (processing level 0) is interrupted and the corresponding STXIT process (processing level 127) is started. When the STXIT process terminates the basic process is continued as long as no other interrupt events have occurred in the meantime. If a further interrupt event has occurred during the STXIT process, the following points apply (depending on the queueing method):

– The current STXIT process is interrupted by a new STXIT process if the latter is classified according to the LIFO principle (nested execution). After the new STXIT process terminates, the interrupted STXIT process is resumed and, when this has terminated, the basic process is continued as long as no new interrupt event has occurred during these runs. It is also possible for more than two STXIT processes to be nested. This also applies if the interrupt event belongs repeatedly to the same event class, in which case a new STXIT process of the same STXIT routine is started. However, this limits the level of nesting (specification in the **STXIT** macro and maximum nesting depth as shown in the table 9 on page 137).

– The current STXIT process terminates and the new STXIT process is started if the latter is classified according to the FIFO principle. The number of consecutive FIFO processes is limited if the interrupt event belongs repeatedly to the same event class (maximum nesting depth in the table 9 on page 137). The basic process is continued if no new interrupt events have occurred during these runs.

For b):

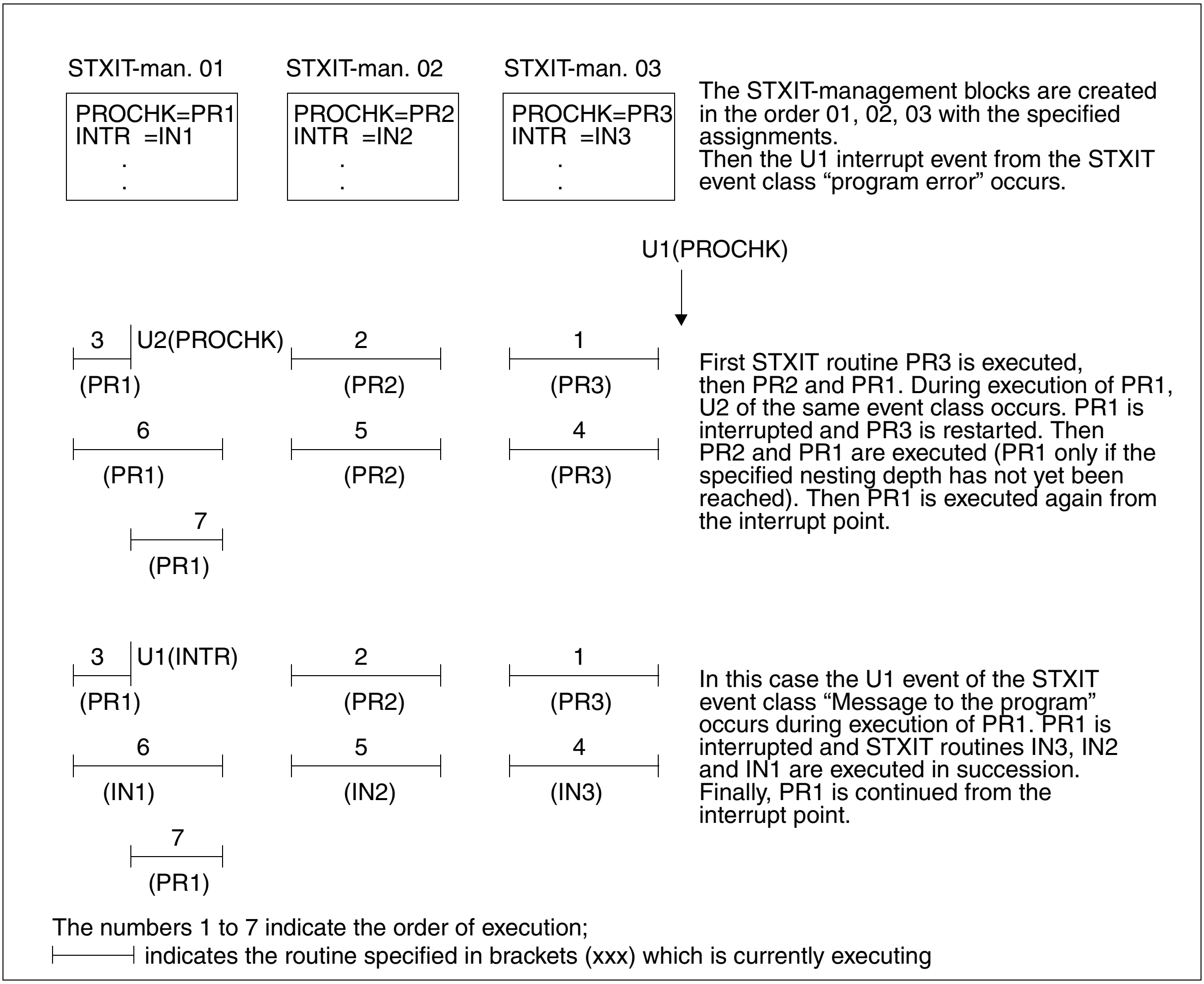
STXIT processes are created for all the STXIT routines of this event class when an interrupt event occurs. These are entered in the process queue according to their specified queueing method, beginning with the first STXIT management block to have been created with this event class. The results of this in a run are as follows:

– For STXIT routines of an event class using the LIFO method:
The basic process is interrupted and the STXIT routine entered in the last STXIT management block to be created is executed, followed by the STXIT routine entered in the previous STXIT management block to be created, etc.

– For STXIT routines of an event class using the FIFO method:
The basic process is interrupted, and the STXIT routine entered in the first STXIT management block to be created is executed, followed by the STXIT routine entered in the next STXIT management block to be created, etc.

- In every STXIT routine the user can specify in the **EXIT** macro whether the process sequence for the same event class is to be interrupted or continued.

If a new interrupt event occurs during an STXIT process, STXIT processes are created (again) for all STXIT routines of the event class in question and entered in the process queue according to the specified queueing method. This also applies if the interrupt event belongs repeatedly to the same event class. However, a new STXIT process is created for an interrupted STXIT routine only if the specified nesting depth has not yet been reached. The figure below illustrates the operation sequence for the LIFO method.



The STXIT-management blocks are created in the order 01, 02, 03 with the specified assignments.
Then the U1 interrupt event from the STXIT event class “program error” occurs.

First STXIT routine PR3 is executed, then PR2 and PR1. During execution of PR1, U2 of the same event class occurs. PR1 is interrupted and PR3 is restarted. Then PR2 and PR1 are executed (PR1 only if the specified nesting depth has not yet been reached). Then PR1 is executed again from the interrupt point.

In this case the U1 event of the STXIT event class “Message to the program” occurs during execution of PR1. PR1 is interrupted and STXIT routines IN3, IN2 and IN1 are executed in succession. Finally, PR1 is continued from the interrupt point.

The numbers 1 to 7 indicate the order of execution;
├────┤ indicates the routine specified in brackets (xxx) which is currently executing

Figure 21: Example of a run

- Control of STXIT processes

  Changing the processing level (process priority):
  As stated above, STXIT processes are started with processing level 127 by default and, if they have the same processing level, they are entered in the process queue according to a specified classification (queueing) principle (LIFO/FIFO). The following applies to the value range:

Basic process:	Processing level 0 - 127; default value 0
STXIT process:	Processing level 1 - 127; default value 127

  The basic process (started with a START-PROGRAM command) has processing level 0 and will always be interrupted by an STXIT process. The processing level (within the specified range) of the STXIT process can be modified with the **LEVCO** macro during execution of the STXIT routine. The user can thus change the order of the active STXIT processes. However, the user must note that the processing level of an active process cannot drop below that of a process which has already been started and interrupted.

  Access to the PCB of the interrupted process:
  The user can obtain read and write access to the PCB of the currently interrupted process or the PCB of the basic process with the **CONTXT** macro. The caller can access the general registers, the floating-point registers and the program counter. Furthermore, the return code informs the user if the PCB has already been modified in the current interrupt state with a **CONTXT** macro.

Restrictions

- Only one STXIT routine can be assigned to the “SVC” event class in one program (program system). The assignment must be carried out in the first **STXIT** macro or refer to the first STXIT management block that was created.
- It must be noted that, for the event classes “timer for real time” and “timer for CPU time”, only one time interval for real time and CPU time is set at any moment (**SETIC** macro).
- Calling the **TERM** macro in the program (program system) leads to the activation of all the STXIT routines assigned to the “TERM” event class, as long as the previous routine was terminated with **EXIT** CONTINU=YES in each case. A further **TERM** macro (also in an STXIT routine) causes immediate program termination, as required.
- When the specified number of nesting levels is exceeded, the program is terminated and a user dump is produced; the interrupt weight in the PCB of the basic task is overwritten with X'06'. (X'06' does not refer to any event to which the user may react by activating a STXIT routine.)

The following table shows the STXIT event classes and their associated interrupt events:

STXIT event class	**Interrupt event**	**Event code**	**Queueing method**	**Maximum nesting depth**
Program error	Invalid SVC Invalid operation code Data error Exponent overflow Divide error or negative square root Mantissa = 0 Exponent underflow Decimal overflow Fixed-point overflow	X'04' X'58' X'60' X'64' X'68' X'6C' X'70' X'74' X'78'	LIFO	127
Interval timer for CPU time	“SETIC interval” expired for CPU time	X'20'	FIFO	127
Interval timer for real time	“SETIC interval” expired for real time summer/wintertime conversion	X'AO' X'CO'	FIFO	127
End of program runtime	CPU time limit for task or program exceeded	X'80'	FIFO	0
Unrecoverable programming error	Privileged SVC Access to a non-existent memory page Privileged operation Address error (e.g. alignment error or wrong register) XA error in SVC macro (24 bit data area used in 31 bit mode) Realtimer (condition error) Data area alignment error in SVC macro Validation error Invalid UNIT no. in standard header	X'08' X'48' X'54' X'5C' X'9C' X'A4' X'AC' X'B0' X'C4'	LIFO	127
Message to the program	INFORM-PROGRAM command	X'44'	LIFO	127
ESCPBRK	BREAK/ESCAPE (via keys)	X'84'	LIFO	127

Table 9: STXIT event classes and their associated interrupt events

STXIT event class	Interrupt event	Event code	Queueing method	Maximum nesting depth
Program terminated by asynchronous events	Error recognized by system, e.g. error in system, loss of power START-PROGRAM, LOAD-PROGRAM, ABEND, EXIT-JOB, CANCEL-JOB Address translation error due to hardware error Hardware error (CPU) Forced unloading of a subsystem (system admin.) Unrecoverable DMS error	X'88' X'8C' X'94' X'A8' X'B8' X'BC'	LIFO	0
Program terminated by synchronous events	TERM-SVC from TU program Program terminated by CMD-/LGOFF macro	X'90' X'98'	LIFO	0
SVC interrupt	SVC macro of a specified SVC	X'50'	LIFO	127
Hardware error	Input/output error when using the “data in virtual” method	X'28'	LIFO	0
Live Migration	Live Migration	X'D0'	FIFO	127

Table 9: STXIT event classes and their associated interrupt events

Example of program structure with STXIT routines in a program system:

The program structure consists of one main program and several subprograms.

An STXIT management block is created by the main program and the STXIT routine “TERMR1” assigned to the “program termination” event class is entered. Another STXIT management block is created by subprogram A and a “TERMR2” STXIT routine is also entered for the “program termination” event class. As soon as an event from this event class occurs, the STXIT routines “TERMR2” and “TERMR1” are executed in succession. The last STXIT management block to have been created is modified later by subprogram A (by referring to the ID of the block); the assignment for the “program termination” event class is deleted and the STXIT routine “ABNDR” assigned to the “ABEND” event class is entered. From this time onward, only the “TERMR1” STXIT routine is activated when an event from the “program termination” event class occurs.

Program structure

```
EXAMP    START               * MAIN PROGRAM
         BALR  ...
         USING ...

***                          * First STXIT call in the main program
         STXIT STXDNEW=STXDIDF1,TERM=(TERMR1)
         :
TERMR1   EQU   *             *  STXIT routine our "program termination"
         BALR  ...
         USING ...
         :
         EXIT
         :
STXIDF1  DC    F             * 4-byte field for the ID of the STXIT
         END                 * management block

         :                   * USUBROUTINE A
         LA    6,STXDIDF2
***                          * Second STXIT call (subroutine A)
         STXIT STXDNEW=(6),TERM=(TERMR2)
         :
***                          * Update for the second STXIT management block.
***                          * The interrupt exit for TERM is closed and
***                          * an STXIT routine is specified for the
***                          * ABEND event class.
         STXIT STXDID=STXIDF2,TERM=(CLOSE),ABEND=(ABNDR)
         :
TERMR2   EQU   *             * STXIT routine for "program termination"
         BALR  ...           * (subroutine A)
         USING ...
         :
         EXIT
         :
ABNDR    EQU   *             * STXIT routine for "ABEND" (subroutine A)
         BALR  ...
         USING ...
         :
         EXIT
         :
STXDIDF2 DC    F             * 4-byte field for the ID of the second STXIT
         END                 * management block
```

4.3.8 Distributed Lock Manager (DLM)

Macro	**Brief description**
LKCAN	Cancels lock requests
LKCVT	Converts lock requests
LKDEQ	Releases lock requests
LKENQ	Generates a lock
LKEQU	Generates DLM-specific layouts
LKINF	Provides information about locks
LKLSB	Generates the layout of the Lock Status Block

The Distributed Lock Manager (DLM) enables tasks running on different nodes of an XCS cluster (XCS = cross-coupled system) to serialize their accesses to shared resources (e.g. files, data records, database blocks, devices, etc.). The resources which are concealed behind a lock are not visible for the DLM: consultation is the responsibility of the users.
The DLM functionality is comparable to the serialization of tasks on a single (local) node (computer). However, it is now also possible to choose between local node locks, i.e. locks which are only visible at the local computer, and cluster locks which are visible at all the computers (= nodes) in the XCS network. To set up an XCS network it is necessary to purchase the product HIPLEX MSCF [26].
In addition, DLM provides a number of lock modes which permit not only a serialization of (exclusive) access but also, for example, parallel, simultaneous reads by different users.

As far as the DLM is concerned, a lock is identified by

– a freely definable lock name assigned by the user
– the local scope (SCOPE operand)
– the global scope (NAMRNGE operand)

These specifications are used to assign the different calling tasks the lock with which they want to work. Only if all three specifications are the same are two tasks referring to the same lock.
To simplify processing, the calling task receives a lock ID which is then used to identify the lock in all subsequent calls.

DLM locks are not linked to locks of any other lock manager. To ensure that a resource is correctly protected, all users of that resource must use the same lock mechanism.

4.3.8.1 Structure of a DLM lock

A lock which is administered by DLM consists of at least two parts:

– The user-independent part that contains the general information and the administrative data for the lock. This exists only once per lock and applies to all users of the lock.

– For each user who works with the lock, there is a user-dependent part which contains information which is specific to each user of the lock. A user-dependent DLM lock part is referred to as a lock request in the following.

All user-dependent lock parts, i.e. all lock requests, are linked to the user-independent part in such a way that together they indicate the status of the individual lock request. The possible statuses of a lock request are dependent on the mode in which the request was made.

● **Lock mode**

The DLM provides six different lock modes:

NU Null mode. A lock request allocated with this mode is compatible with all other lock requests. However, it may not access the resources.

CR Concurrent-Read mode. The lock holder is granted unprotected read access to the resource, i.e. other read or write accesses to this resource are permitted at the same time. Other lock holders are permitted only in NU mode, CR mode, CW mode, PR mode and PW mode.

CW Concurrent-Write mode. The lock holder is granted unprotected write access to the resource, i.e. other read or write accesses to this resource are permitted at the same time. Other lock holders are permitted only in NU mode, CW mode or CR mode.

PR Protected-Read mode. The lock holder is granted protected read access to the resource, i.e. no other read or write accesses to this resource are permitted at the same time. Other lock holders are permitted only in NU mode, CR mode or PR mode.

PW Protected-Write mode. The lock holder is granted protected write access to the resource. Other lock holders are permitted only in CR mode and in NU mode.

EX Exclusive mode. The resource may only be accessed by the lock holder. No other read or write accesses are permitted. Lock holders in NU mode are compatible with the EX mode, but they may not access the resource.

- **Compatibility of lock modes**

Weak
↑
Lock protection
↓
Strong

Strongest currently allocated lock mode	Requested lock mode					
	NU 1	CR 2	CW 3	PR 4	PW 5	EX 6
NU 1	+	+	+	+	+	+
CR 2	+	+	+	+	+	-
CW 3	+	+	+	-	-	-
PR 4	+	+	-	+	-	-
PW 5	+	+	-	-	-	-
EX 6	+	-	-	-	-	-

+ requested lock mode is compatible with the allocated lock mode
- requested lock mode is incompatible with the allocated lock mode

- **Status of a lock request**

A lock request may have one of the following statuses (as seen by the user):

GRANTED The lock request is allocated for a lock if it is compatible with all other already allocated locks. New lock requests are only allocated if they are compatible or if the allocated lock request has been released.

CONVERTING The lock request has already been allocated in one mode but this mode now has to be changed. If this conversion leads to incompatibilities with other allocated lock requests, it must be postponed until the new lock request can be compatibly allocated. The lock request to be converted remains in its original lock mode until it can be converted (or until it times out or is cancelled).

WAITING The lock is waiting for its initial allocation. Allocation is performed as soon as it is compatible with all the other lock requests that have already been allocated

Lock requests with the status CONVERTING are handled before any other new lock requests with the status WAITING.

If there is a lock request with the status WAITING which is incompatible with the currently allocated lock requests, it blocks all subsequent lock requests for this lock even if these are compatible (apart from locks in NU mode).

For questions of lock mode compatibility, see the table above.

- **Lock Value Block**

  The Lock Value Block (LVB) is a small address space which is directly allocated to the lock. It can be read or written depending on the mode in which the lock request is allocated and whether the lock request is receiving (strong lock mode) or releasing (weak lock mode) the lock. The LVB enables users of the lock to exchange information with one another.

  The LVB consists of one 16-byte area (lock value) which can be described by the user with any information. The lock value which the user wishes to write or which is supplied to the user during a read request is located in the Lock Status Block.

  The LVB exists for as long as any lock request exists for that lock. When the last (or only) lock request for the lock is released, the lock itself is canceled and the LVB discarded.

  The Lock Value Block is accessed via the macros **LKENQ**, **LKCVT** and **LKDEQ**.

  Accesses to the Lock Value Block

from the allocated lock mode	to the requested lock mode					
	NU	CR	CW	PR	PW	EX
NU	r	r	r	r	r	r
CR	-	r	r	r	r	r
CW	-	-	r	r	r	r
PR	-	-	-	r	r	r
PW	w	w	w	w	w	r
EX	w	w	w	w	w	w

-	The lock value is neither read nor written.
r (read)	The lock value is read. The lock value can be read by the user when the lock request is allocated. If the lock value of a lock is read for the first time, it contains the initialization value (binary nulls).
w (write)	The lock value is written. The lock value can be written by the user when the lock is converted from a stronger lock mode (e.g. PW or EX mode) to a weaker or equal lock mode or while a lock in PW or EX mode is being released.

If it was specified in the **LKENQ** macro that the LVB is to be read, the request is treated as if the lock were to be converted from NU mode to the specified lock mode.

The illustration below indicates the diagrammatic structure of the resulting lock.

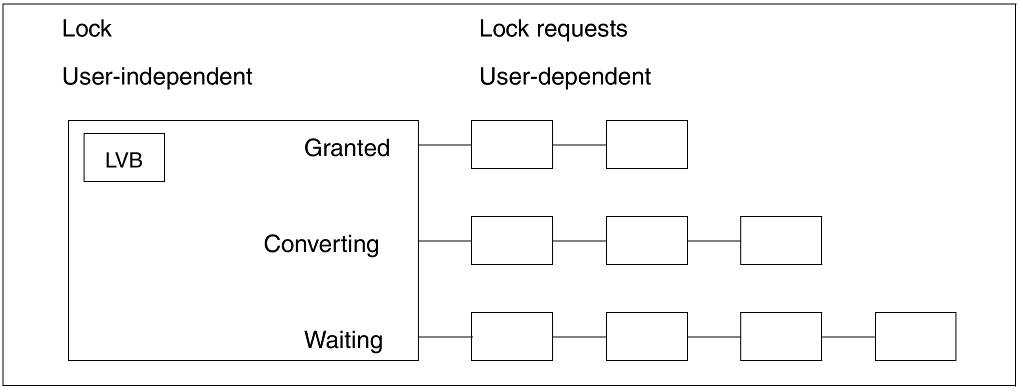


Figure 22: Schematic diagram of a lock

4.3.8.2 Functions of the DLM

- **Generate lock**

  A task wishing to use a lock must first generate it. This is performed implicitly by calling the **LKENQ** macro. The lock is generated by the DLM.
  When a new lock that does not yet exist is generated, both parts of the lock, namely the user-dependent and the user-independent parts, are generated.
  If the lock to be generated already exists, only the user-dependent part of the lock is generated and concatenated with the user-independent part.

  Lock ID (LOCKID)

  After a lock is generated, the task is given a (task-specific) lock ID (LOCKID) (**LKENQ** macro, LOCKID operand). The lock ID must be specified by the calling task if other calls referring to the same lock are made to the DLM. Other tasks that use the same lock and thus the same lock name have different lock IDs for that lock.
  The lock ID becomes invalid when the calling program releases it (**LKDEQ** macro) or when the program is terminated.
  A value equal to 0 is never supplied by the DLM as a lock ID.

  i To avoid incompatible lock requests, a new lock request must always be generated with LCKMODE = *NU (**LKENQ** macro) so that it can be allocated immediately (GRANTED status). Otherwise the lock request receives the WAITING status which it retains until all lock requests with the status GRANTED and CONVERTING have been released before it can be allocated. When the lock request has been allocated the lock mode can be converted with the **LKCVT** macro, whereupon the request switches between the GRANTED and the CONVERTING statuses only (see page 142).

- **Lock request conversion**

  An existing and already allocated lock belonging to a user can be converted from one lock mode to another (**LKCVT** macro). The **LKCVT** requests are executed before a new lock request (**LKENQ** macro), apart from lock requests in NU mode, is executed.

  During the conversion of the lock mode, the lock remains allocated in the original lock mode. The lock request receives the CONVERTING status. If the lock request can be allocated, then it is assigned the GRANTED status.The lock mode has now been converted.

Relationship between lock modes

Lock mode1	Lock mode 2					
	NU	CR	CW	PR	PW	EX
NU	e	h	h	h	h	h
CR	l	e	h	h	h	h
CW	l	l	e	h	h	h
PR	l	l	h	e	h	h
PW	l	l	l	l	e	h
EX	l	l	l	l	l	e

e Lock mode 2 is equal to lock mode 1
l Lock mode 2 is weaker than lock mode 1
h Lock mode 2 is stronger than lock mode 1

- **Releasing a lock**

  A lock can be released in two ways:

  1. Firstly, the lock can be released by the calling task (**LKDEQ** macro).
     This means that the caller's allocated lock request is canceled and the lock ID (LOCKID) is invalidated. Further use of the invalid lock ID leads to error codes, particularly if the lock still exists.
     The user-dependent part of the lock, and all its user-specific information which has been stored together with the lock, are deleted from the DLM data structure. If the user is the last (only) user of this lock, the user-independent part of the lock is also deleted. In this way the lock itself is deleted from the DLM data structure.

  2. Secondly, a task's existing lock request can be converted to a weaker lock mode (see table on page 142). The lock ID (LOCKID) remains valid and can continue to be used (**LKCVT** macro).

- **Canceling a lock request**

  Lock requests which have not yet been allocated by the DLM can be canceled with the **LKCAN** macro.

  Lock requests which have been generated by the **LKENQ** macro and have the status WAITING or CONVERTING are completely canceled. If the lock request has the status GRANTED, i.e. if the lock is already allocated, processing is terminated with an error code. This lock request must be released with the **LKDEQ** macro.

  Lock requests which are to be converted with the **LKCVT** macro and have the status CONVERTING are not canceled. Only the conversion job is canceled.

  The call can be synchronous or asynchronous.

- **Outputting information on locks**

  The **LKINF** macro provides information on which locks are already being used. Several search filters can be activated to narrow the selection. The locks are identified by their lock names. The lock names may be fully or partially qualified names. If partially qualified lock names are specified, the access operation can take a very long time as the entire DLM database has to be searched for hits. This depends on the size of the DLM database.

- **Timeout detection**

  The user can specify a wait time and a hold time for the lock request, which is checked by the DLM. If the lock request cannot be allocated during the wait time, it is terminated with an error code or a wait time timeout is generated.

  If the hold time exceeds the time limit, the lock holder is informed by the release event. If a hold time is specified, a release event (RELEVTT) must also be specified.

  If the value 0 was entered as the time limit for the lock request, this lock request is identified as an immediate request. Immediate requests which cannot be allocated are terminated with the return code X'00828006'.

- **Lock Status Block**

  The Lock Status Block (LSB) is part of the user address space and has two main tasks.

  Firstly, the LSB is needed for all asynchronous DLM calls. The LSB is the communication area between the DLM and the calling program. The asynchronous allocation of the lock is notified to the user by the appropriate return code being set in the LSB. The user is informed of the specified event method.

  The LSB must be initialized before asynchronous functions can be requested by the DLM. This is done by calling the **LKLSB** macro with MF=L. The initialization values are written in the LSV area. The DLM can now decide whether or not the transferred address indicates a valid LSB. If the lock is allocated, the LSB must be available for the DLM, otherwise no data is transferred. The error is reported to the calling program using the event method.

  Secondly, the LSB is needed if the Lock Value Block is to be read or written, irrespective of whether the requests are synchronous or asynchronous.

  The lock value which the user wishes to write or which is supplied to the user during a read request is located in the Lock Status Block. The address of the LSB is specified in the operand list of the current DLM call (LSBADR operand in the macros **LKENQ**, **LKCVT**, **LKDEQ** and **LKCAN**).

- **Status information**

  The DLM keeps status information about the release of the last lock in PW or EX mode. The release was executed normally if the lock was released by the lock holder (**LKDEQ** macro). The status information is set to VALID.

  If the release was terminated abnormally, the status information is set to INVALID. The release is terminated abnormally if the program, the task, or the node on which the lock holder is located was terminated before the lock was released.

  The status information provides information about the validity of the Lock Value Block, which is handled in the same way. The handling of the status information is not necessarily linked to the handling of the Lock Value Block

  The status information is sent to every subsequent lock holder until the status information has been reset to VALID. If the requested lock mode was NU or CR the information may no longer be current.

  The status information INVALID can be reset to VALID if another lock holder requests the lock in PW or EX mode, is allocated it, and, when it is released, specifies that the status is to be reset. This can take place irrespective of the Lock Value Block. However, it is possible to change the Lock Value Block and reset the status to VALID with a DLM call.

- **Termination sequence of lock requests during termination of the lock holder process**

  During abnormal termination of a lock holder process (task or program), the lock requests are released in a defined sequence.

  The caller can specify the release time for his lock in one of the following three classes during generation of a lock request (TERMNTE operand in the **LKENQ** macro):

  FIRST This class contains locks which are released before or at the same time as the locks in the next class, SECOND.

  SECOND This class contains locks which are released after or at the same time as the locks in the previous class, FIRST, and before or at the same time as the locks in the next class, THIRD

  THIRD This class contains locks which are released after or at the same time as the locks in the previous class, SECOND.

  To ensure that all locks from different processes which belong to the same application are handled in the same way, the caller must ensure that all **LKENQ** requests are called with the same operands.
  This is not checked by the DLM during the **LKENQ** call.

4.3.8.3 Synchronous and asynchronous lock requests

The DLM functions can be executed in two ways:

Firstly, the call can be synchronous. This means that control is only returned to the caller when the request has been allocated or an error condition detected. The information is returned in the return code.

Secondly, the call can be asynchronous. This means that control is returned to the caller when the request has been accepted by the DLM. The lock request is allocated later. To this end, the caller must specify an event method.

Two methods are possible for the asynchronous event method: contingency process or eventing.
It is possible for one user to use the synchronous method and another user an asynchronous method for the same lock and at the same time.

The following table shows which operands can be specified in the **LKENQ** macro and the **LKCVT** macro in order to use the desired event method.

Operands in the **LKENQ** and **LKCVT** macros

Event method		GRTEVTT	RELEVTT
Synchronous allocation		*SYNCH	*NO
		*SYNCH	*TUCONTI
		*SYNCH	*TUEVENT
Asynchronous allocation	Contingency	*TUCONTI	*TUCONTI
		*TUCONTI	*TUEVENT
	Eventing	*TUEVENT	*TUCONTI
		*TUEVENT	*TUEVENT

Support overview:

Macro	**Synchronous**	**Asynchronous**
LKENQ	x	x
LKCVT	x	x
LKDEQ	x	x
LKCAN	x	x
LKINF	x	-

- **Events and event specification**

  The DLM provides:

  – the allocation event (GRANTID)
  – the release event (RELID)

  The allocation event terminates an asynchronous lock request either after the lock has been allocated or when an error situation has been detected.

  The release event informs the lock holder that his own allocated lock is blocking incompatible lock requests from other users. The lock mode of the blocked request is transferred together with the event. The lock holder is now able to reduce his lock protection (**LKCVT** macro) or release his lock (**LKDEQ** macro).

  If a lock holder releases the lock and the new lock holder blocks other lock requests, the release event (which is sent to a blocking lock request) is also generated for the new blocking lock request. The user must therefore take into account release events which may be sent immediately after the lock is allocated. It is also possible for a lock holder which is blocking other lock requests to receive more than one release event.

  If an asynchronous DLM call times out, this causes a waiting-time timeout event.

  If a lock is held for longer than the hold time specified for the lock, this causes a hold-time timeout event.

  – *Asynchronous events and USERPAR operand*
    The asynchronous functions enable the USERPAR operand to be specified. The value which the user specified when calling the macro is returned in the concluding event of this lock request.

    The USERPAR value which was specified in the **LKENQ** macro is transferred together with the event when an allocation or release event for this lock (and this task) occurs.
    As of now, the USERPAR value transferred in a subsequent **LKCVT** macro is used by the DLM for eventing. Every subsequent allocation or release event is transferred together with this USERPAR value.

    The USERPAR value which was specified in the **LKCAN** or **LKDEQ** macro is used only for confirmation of the cancellation event or of the release event.

    For ease of use, the user should specify only one USERPAR value for all macros which relate to the same lock request.

  – *Synchronous*
    The user does not regain control until the lock request has been allocated, the waiting time has expired or an error condition has been detected. The corresponding information is provided in the return code.

- *Contingency process*
  A contingency ID can be specified for each of the events. The allocation event and the release event may specify different contingency processes. If one of the two events occurs, the corresponding contingency process is provided. If the release event occurs, the blocking lock mode is transferred to the contingency process.

  The contingency process must be generated with the **ENACO** macro. The returned contingency ID must be specified in every asynchronous call. The contingency process is provided under the control of the lock requester task.
  The contingency ID must be available for the DLM, as otherwise no event can be supplied.

  When the contingency process is started, it is sent information about the asynchronous call. This information consists of the event that caused thecontingency process to be provided. It is transferred to the contingency process via registers. Register 3 contains the event specification; register 4 contains the user-defined USERPAR values from the **LKENQ**, **LKCVT** or **LKDEQ** macro. The status of the asynchronous call is shown in the Lock Status Block which was specified in the DLM call.

  In order to use the contingency process it is necessary to specify two contingency IDs: one for the allocation event (GRANTID) and one for the release event (RELID). The ID can be the same in both cases. The decision as to which event is reported can be made with the help of the information transferred to the contingency process.

- *Eventing*
  It is advisable to specify the same event ID for both events. The event ID must be generated with the **ENAEI** macro. The returned EIID must be specified for every asynchronous call.
  The event ID must be available at the moment when the asynchronous request is allocated as otherwise no event is supplied.

  The postcode, which is transferred together with the event, contains the event specification (DLM event). The user-defined operand USERPAR is also transferred in the second word of the postcode. Information about the asynchronous request, if available, is also transferred to the Lock Status Block.

- *(Eventing) and contingency process*
  A combination of the two methods is possible. See the table “Operands in the LKENQ and LKCVT macros” on page 149.

4.3.8.4 Lock name

The lock name is a unique identifier for the lock. As far as the DLM is concerned, a lock name internally consists of three parts (IDs for the local and global scope and a string for the lock name). From the user's point of view, the following operands affect the way the internal lock name is formed (**LKENQ** and **LKINF** macros).

1. The NAMRNGE operand is used to specify the global scope of the lock.
   - NAMRNGE=*OWNSYSTEM
     The specified lock name is only valid on the local system.
   - NAMRNGE=*CLUSTER
     The specified lock name is valid throughout the cluster.
2. The SCOPE operator is used to specify the local scope of the lock.
   - SCOPE=*NAMESPACEID
     The specified lock name is used as the internal lock name. The first part (8 bytes) of the specified lock name implicitly forms the local scope. The local scope must be a string. Valid characters are the letters “A..Z”, “a..z”; the digits “0..9” and the special characters “@” and “#”. The maximum length of the lock name is 48 characters.
   - SCOPE=*USERID
     The user ID to which the calling task belongs is used to form the internal lock name. The DLM determines the user ID and places it at the start of the lock name. The first part of the specified lock name is not considered to be the local scope. The maximum length of the specified lock name is reduced to 40 characters.
     Specifying SCOPE=*USERID is an easy way of protecting an application's locks against access by another application. The applications simply have to be started under different user IDs.
   - SCOPE=*GROUPID
     The user group to which the calling task belongs is used to form the internal lock name. The DLM determines the user group and places it at the start of the lock name. The first part of the specified lock name is not considered to be the local scope. The maximum length of the specified lock name is reduced to 40 characters.
     The SCOPE=*GROUPID operand may only be specified if the software product SECOS has been purchased and is operational as otherwise the **LKENQ** call results in an error.
     Specifying this operand is an easy way of protecting an application's locks against access by another application. The applications simply have to be started under different user groups.
3. The remaining part of a lock name (up to 40 characters) may contain any characters.

The local scope must be formed in accordance with the following rules:

1. Users are not permitted to form a local scope which starts with the special character “$”.
2. Users must avoid the prefixes “SYS” and “@” for the local scope. These prefixes are reserved for internal TU applications.

4.3.8.5 Cluster systems and single systems

Every lock name belongs to a global scope. If this scope is local to the node, the name is valid only on its own node. It is always separate from a name which is the same but which has the cluster as its scope.

Two applications (running on different nodes in a cluster) that have specified the same lock name with local scope are not serialized because each of the lock names is a local, node-specific name.

If two applications specify the cluster as the scope for identical lock names, they refer to the same lock and are serialized by the DLM.

If two applications (running anywhere on the cluster) specify the same lock names but different scopes, they refer to different locks.

A lock with NAMRNGE=*OWNSYSTEM cannot be transferred to a lock with NAMRNGE=*CLUSTER because these two locks are completely different.

To enable user programs (that were developed for cluster systems) to run on single systems without the need for modifications, the DLM provides the following feature:

If a system is not part of a cluster and is also not part of a cluster in the current session, the DLM accepts **LKENQ** calls as if this system were part of a cluster. Locks which have been enqueued with NAMRNGE=*CLUSTER are not the same as those enqueued with NAMRNGE=*OWNSYSTEM.

Brief overview

Specify when calling macro	Macro	Returned by macro	
MF=D	LKEQU		(1)
MF=L	LKLSB		(2)
	LKINF		(3)
	LKENQ	LOCKID	(4)
LOCKID	LKCVT		(5)
LOCKID	LKCAN		(6)
LOCKID	LKDEQ		(7)

(1) **LKEQU** generates DLM-specific layouts

(2) **LKLSB** generates the layout of the Lock Status Block

(3) **LKINF** provides information about locks

(4) **LKENQ** creates a lock and supplies a lock ID

(5) **LKCVT** converts the lock request

(6) **LKCAN** cancels the lock request

(7) **LKDEQ** releases lock requests

4.4 Requesting and accessing lists and tables

Macro	**Brief description**
AINF	Provides details of the job' s resource utilization
CHKPRV	Checks the current job for privileges
CTIME	Performs time stamp calculations (represents different formats and works with timespans)
CUPAB	Generates addressing aids (DSECTs) for the operand lists of the RDATA, WROUT and WRTRD macros
DCSTA	Generates an area (CSECT) or addressing aids (DSECTs) for the information supplied by the TSTAT macro
DJINF	Generates a DSECT/data list for the output area of the JINF macro
DTMODE	Generates a DSECT/data list for the 31-bit interface of the TMODE macro
GCCSN	Displays the current code table for input and output of command and data (SYSDTA/SYSCMD/SYSOUT/SYSLST)
GEPRT	Gives the elapsed CPU time and the CPU time still available to the program or job
GTIME	Gives the current date and time, and gives information about the current time zone
HSITYPE	Supplies information about the current hardware-software interface (HSI)
IOSID	Supplies information on the ID and version number of the operating system
JINF	Transfers a list of job data (SET-LOGON-PARAMETERS command / ENTER-JOB command) relating to the job executing the calling program
JMGDJP	Generates a DSECT/data list for the data area of the JMGJPAR macro
JMGJPAR	Transfers the job parameters specified in the SET-LOGON-PARAMETERS command or ENTER-JOB command)
JOBINFO	Transfers a list of job data (SET-LOGON-PARAMETERS command / ENTER-JOB command) for a specified job
MINF	Outputs memory map and size for class 6 memory or memory pool
NKDINF	Transfers information on the reservation and availability states of the (peripheral) configuration
NKGTYPE	Supplies the name, device type code, device attributes, etc. of a device/volume type or the names and device type codes of the device types that belong to a device family or device class
NSIINF	Supplies information on CPU, operating system, HSI or memory
NSIOPT	Supplies information on system parameters
RDUID	Transfers to the program the user ID of the task under which it is running
SINF	Supplies information on CPU, operating system or system parameters to a user program area
SRMUINF	Transfers data from the user catalog into an area

Macro	**Brief description**
STAMCE	Reads entries from the MRSCAT
TMODE	Supplies information on the job currently running, e.g. TSN, user ID, accounting number, etc.
TSPRIO	Generates symbolic names for upper and lower limits of variable and constant priorities
TSTAT	Supplies information on the characteristics of the interactive terminal (see also DCSTA)
VMGINF	Supplies information in connection with VM2000 operation
VTCSET	Generates symbolic names used to insert logical control characters in line mode output messages or to locate line mode inputs

4.5 Input/output

4.5.1 System files

Macro	**Brief description**
SYSFL	Reassigns the system files SYSDTA, SYSLST, SYSLST01, SYSLST02, ..., SYSLST99 and SYSOUT and the TASKLIB.
SYSTA	Assigns the system files and the TASKLIB.

The standard file names SYSDTA, SYSCMD, SYSLST, SYSLST01, SYSLST02, ..., SYSLST99 and SYSOUT denote (system) files used by the operating system for the input of commands and data to the operating system, or for the output of data by the operating system. These files are created by the user task and define initial (primary) default input and output areas.
Users can revoke the primary assignment and assign their own (cataloged) files to the (standard) file names. Some of the standard names can also be equated (see the **SYSFL** macro).

i The system files SYSIPT and SYSOPT are supported only for reasons of compatibility. They are no longer described in this manual.

A task can use the following system files for input

SYSCMD The commands used to control the job are read in from SYSCMD.

SYSDTA The SYSDTA system file is used to input data and instructions for a program. As soon as a program executes, SYSDTA is active. SYSDTA can be accessed by means of the **RDATA** macro.

A task can use the following system files for output

SYSOUT All logged messages and error messages that occur during the current job are written to the SYSOUT system file. Utility routines and compilers also use SYSOUT for this purpose. SYSOUT can be accessed by means of the **WROUT** macro.

SYSLST SYSLST is used primarily for storing larger amounts of data, e.g. dumps or generated lists. SYSLST can be accessed by means of the **WRLST** macro. In addition, any data records written to SYSOUT are also written to the system file SYSLST if the appropriate operands were specified in the SET-LOGON-PARAMETERS or MODIFY-JOB-OPTIONS command.

SYSLST01, SYSLST02,...,SYSLST99
Unlike the SYSLST system file, these files do not have their own EAM area for storing output data. They are used for intermediate storage and are effective only if they are assigned cataloged files. The SYSLSTn files can be accessed by means of the **WRLST** macro.

System files for output will be created as necessary by the operating system under the user's ID. These are SAM files with the file names

S.OUT.tsn.yyyy-mm-dd.hhmmss (for SYSOUT)
S.LST.tsn.yyyy-mm-dd.hhmmss (for SYSLST)

where:

- tsn=TSN (task sequence number) assigned to the job
- yyyy-mm-dd=Date (yyyy=year, mm=month, dd=day)
- hhmmss=Time (hh=hours, mm=minutes, ss=seconds)

The storage space occupied by the files is not counted as part of the public space allocation. At the end of the job, the files are automatically printed out, and then deleted. The user has no access to these files. The command /DELETE-FILE *SYSxyz (xyz=LST/OUT/OPT) may be used to (logically) delete the contents of a specified system file, although the catalog entry is retained. An empty system file will not be printed out.

The system files used for output may also be output earlier than normal (START-PROCESSING operand in the PRINT-DOCUMENTcommand).
In the PRINT-DOCUMENT and DELETE-SYSTEM-FILE commands or the **ERASE** (see “DMS Macros” manual [7]) and **PRNT...** macros (see “SPOOL & Print - Macros and Exits” manual [23]), it is permissible to specify the (standard) file namesSYSOUT and SYSLST even if cataloged files have been assigned tothem.

Primary assignment and reassignment of system files

System files are generally given a specific (standard) assignment. This primary assignment can be modified with the aid of the commands summarized in the following table. Examples are to be found in the relevant command descriptions in the “Commands” manual [19]).

System file	**System file assignment Primary assignment**	Other assignments	**Commands for changing the file assignment**
SYSCMD	Interactive mode: terminal Batch mode: spoolin file “S.INtsn” (spooled in via magnetic tape device or ENTER file)	Cataloged disk file (SAM/ISAM)	CALL-PROCEDURE command: assignment to a cataloged file END-PROCEDURE commands (procedure files only) and EXIT-PROCEDURE: return to the last procedure step left via a CALL-PROCEDURE
SYSDTA	See SYSCMD primary assignment	Cataloged disk file (SAM/ISAM), S variable or PLAM library element	ASSIGN-SYSDTA command: assignment to cataloged file, S variable, PLAM library element, SYSCMD, or return to primary assignment. END-PROCEDURE commands (procedure files only) and EXIT-PROCEDURE: return to assignment valid prior to invocation of procedure mode
SYSOUT	Interactive mode: terminal Batch mode: temporary (system) file S.OUT.. which is output to printer at job termination and then deleted	Interactive mode: cataloged file, S variable or PLAM library element Batch mode: cataloged file, S variable or PLAM library element which is not automatically output to printer; PRINT-FILE command required	See SYSLST

Table 10: System file assignment (Teil 1 von 2)

System file	**System file assignment Primary assignment**	Other assignments	**Commands for changing the file assignment**
SYSLST	Temporary (system) file S.LST.... which is output to printer when the job is terminated and then deleted (created only when required)	Cataloged file, S variable or PLAM library element which is not automatically output to printer; PRINT-FILE command required	ASSIGN-SYSLST command: assignment to a cataloged file, S variable or PLAM library element, or return to primary assignment. END-PROCEDURE commands (procedure files only) and EXIT-PROCEDURE: return to assignment valid prior to invocation of procedure mode
SYSLST01 . . . SYSLST99	System files: primary assignment = same as for SYSLST	See SYSLST; also possible between each other	See SYSLST

Table 10: System file assignment (Teil 2 von 2)

4.5.2 Files and records

Macro	**Brief description**
CUPAB	Generates addressing aids (DSECTs) for the operand lists of the RDATA, WROUT and WRTRD macros
GCCSN	Displays the current coding table for command and data input (SYSDTA/SYSCMD)
RDATA	Reads a record from the SYSDTA system file, i.e. from a cataloged file, an S variable, a PLAM library element or the interactive terminal (see also CUPAB)
VTSUCB	Creates VTSU parameters for input/output
WRLST	Causes a record to be written to the SYSLST system file and/or the SYSLST01, SYSLST02, ... ,SYSLST99 system files. The relevant system file is printed after the job has terminated, provided the user has not directed it to a cataloged file
WROUT	Causes a record to be written to the SYSOUT system file, i.e. in interactive mode to the terminal, a cataloged file, an S variable or a PLAM library element. In batch mode the SYSOUT system file is printed after job termination (see also CUPAB), unless the user has assigned it to a cataloged file, an S variable or a PLAM library element
WRTRD	In interactive mode, writes a message to the terminal and subsequently accepts a message from there (see also CUPAB)

4.5.3 Data terminal communication

Macro	Brief description
CUPAB	Generates addressing aids (DSECTs) for the operand lists of the RDATA, WROUT and WRTRD macros
GCCSN	Displays the current code table for input and output of command and data (SYSDTA/SYSCMD/SYSOUT/SYSLST)
DCSTA	Generates an area (CSECT) or addressing aids (DSECTs) for the information supplied by the TSTAT macro
RDATA	Reads a record from the SYSDTA system file, i.e. from a cataloged file, an S variable, a PLAM library element or the interactive terminal (see also CUPAB)
SETBF	Changes the size of the system input/output buffer of the terminal
TCHNG	Determines whether screen overflow is controlled by the system or by the user program
TMODE	Supplies information on the calling task, e.g. buffer size, line length and logical terminal type
TSTAT	Supplies information on the features of the interactive terminal (see also DCSTA)
TYPIO	Outputs a message at the console and accepts a response from the operator
VTCSET	Defines logical control characters
VTSUCB	Generates VTSU parameters for input/output
WROUT	Writes a record to the SYSOUT system file, i.e. in interactive mode to the terminal, a cataloged file, an S variable or a PLAM library element. In batch mode the SYSOUT system file is printed after job termination (see also CUPAB), unless the user has assigned it to a cataloged file, an S variable or a PLAM library element
WRTRD	In interactive mode, writes a message to the terminal and subsequently accepts a message from there (see also CUPAB)

4.5.4 Messages

Macro	**Brief description**
MSG7	Writes a message (with a 7-character message code) to the SYSOUT system file or to the console and accepts a response
MSG7X	Writes a message (with a 7-character message code) to the SYSOUT system file or to the console and accepts a response
MSGSHOW	Supplies information about the scope, number, language, names and access methods of message files
MSGSINIT	System administration macro. This macro modifies the global range assignment list by entering new message files or locking access to message files
MSGSMOD	System administration and user macro. The macro is used to modify the global range assignment list: it enters new message files and/or locks access to message files. The nonprivileged user can add task-specific message files to the messages
MSGRC	Outputs the return code with an explanation for the macros of the messages
OPSGEN	Supplies MIP with information about S variable generation
TYPIO	Outputs a message on the operator console, and accepts a reply

4.5.5 Encryption

Macro	**Brief description**
CRYPT	Encrypts words using a one-way encryption procedure (decryption is not possible).

4.6 Debugging aids

Macro	Brief description
AUDIT	Monitors processor states TU and TPR of the user program
BKPT	Transfers control to the system. The user may then enter commands at the data display terminal
CDUMP2	Initiates a memory dump without program termination
TERM	Terminates program and job step and initiates (where necessary) a memory dump

4.7 Checkpoints

Macro	Brief description
WRCPT	Writes a checkpoint. Meaning: a defined system and program status at a specific point of the program execution is buffered and is available for restarting the program at this point (by means of the RESTART-PROGRAM command)

4.8 Accounting

Macro	Brief description
ARDS	Describes the structure of the user records (definition macro)
AREC	Writes an accounting record to the accounting file
ASPC	Used to record the memory space currently occupied on public volumes (only for system administration)

4.9 Communication (programs, users, system)

Macro	Brief description
BKPT	Transfers control to the system. The user may then enter commands at the data display terminal
CLCOM	Terminates intertask communication for the calling program (see page 76)
CMD	Executes commands without leaving program mode
JINF	Transfers a list of job data
OPCOM	Opens intertask communication for the calling program and defines its ITC name (see page 76)
RELBF	Deletes the first message in the ITC queue of the calling program (see page 76)
REVNT	Receives a message for the ITC name of the calling program (see page 76)
SEVNT	Sends a message to an ITC user (see page 76)
STXIT	Specifies user routines for interrupt handling which are to be started if a program interrupt occurs. By means of the INFORM-PROGRAM command data can be transferred to an interrupt routine (see the “ Commands” manual [19])
SWITCH	Switches and transfers the 32 job switches associated with the job or the 32 user switches associated with the user' s own or another user' s user ID
TMODE	Transfers job attributes such as characteristics of the data display terminal (type, buffer, mode, ...) as well as run priority, TSN, CPU time, and user ID
TYPIO	Outputs a message on the console and accepts a response
WROUT	Transfers a record to the SYSOUT system file, i.e. in interactive mode to the data display terminal. In batch mode, the system file SYSOUT is printed after job termination (see also CUPAB) unless the user has assigned it to a cataloged file
WRTRD	In interactive mode, transfers a message to the data display terminal and subsequently accepts a message from there (see also CUPAB)

The **SWITCH** macro replaces the **GETSW**, **GETUS**, **SETSW** and **SETUS** macros. These macros are still supported for reasons of compatibility only and are described in the appendix on page 1107.

4.10 Multiprocessor systems

Macro	Brief description
MCSINFO	Displays the current HIPLEX MSCF configuration to which the computer belongs
MRSINF	Outputs information on the MSCF communication network
MRSSTA	Outputs status in MSCF
STAMCE	Transfers MRSCAT entries to an area

The macro **MCSINFO** replaces both the **MRSINF** and **MRSSTA** macros. These macros are still supported for compatibility reasons only and are described in the appendix on page 1112. The macro **MCSINFO** is described in the “HIPLEX MSCF” manual [26].

The macros **MCSINFO** and **MRSSTA** are only available to users of the multiuser system, details of which are contained in the “HIPLEX MSCF” manual [26].

4.11 XS programming

Macro	Brief description
AMODE31	States whether the 31-bit addressing mode has been activated
GPARMOD	Determines which interface (24-bit or 31-bit interface) is to be generated for the subsequent macros
HSITYPE	Indicates the size of the addressable class 6 memory and states which addressing mode (24-bit or 31-bit addressing) can be used

4.12 Job scheduler

Macro	**Brief description**
DJSI	Definition macro; creates name definitions, DSECTs or I/O areas for the 24-bit interface of the job scheduler macros
DJSIPL	Definition macro; creates name definitions, DSECTs or I/O areas for the 31-bit interface of the job scheduler macros
JSATTCH	Connects the job scheduler to the job management system
JSDETCH	Severs the connection between the job scheduler and the job management system
JSEXPCT	Requests the next event for the job scheduler from the job management system
JSINFO	Transfers the STREAM-PARAMETER (S-PAR) of the stream definition to an area to be specified
JSRUNJB	Calls upon the class scheduler to start the specified job
JSWAKE	Initiates a timer event for the job scheduler

These macros enable a job scheduler developed by the user to be connected to the Job Management System (JMS).
A job scheduler runs as an application program in the TU processor state and is therefore easily exchangeable. It communicates with JMS via a privileged interface (job scheduler interface).
Users can replace the standard scheduler by a scheduler they have developed themselves to suit their own requirements, without having to intervene in the operating system.
However, the functional scope of this type of scheduler is limited by the functional width of the job scheduler interface.

4.13 Macros generating only CSECTs or DSECTs

Macro	Brief description
ARDS	Describes the structure of accounting records
CUPAB	Generates addressing aids for the operand tables of the RDATA, WROUT and WRTRD macros
DCSTA	Generates an area (CSECT) or addressing aids (DSECT) for the information transferred by the TSTAT macro
DJINF	Generates a DSECT/data list for the output area of the JINF macro
DJSI	Generates name definitions, DSECTs or data areas for the 24-bit interface of the job scheduler macros
DJSIPL	Generates name definitions, DSECTs or data areas for the 31-bit interface of the job scheduler macros
DTMODE	Generates a DSECT/data list for the 31-bit interface of the TMODE macro
JMGDJP	Generates a DSECT/data list for the data area of the JMGJPAR macro
LKEQU	Generates a DSECT/data list for the Event Type Codes and the Global Return Codes that are set by the different macros of the DLM
LKLSB	Generates the layout of the Lock Status Block
PBTABD	Generates a DSECT/data list for the input table of the TABLE macro (for 31-bit interface only). See page 1138
VTCSET	Generates symbolic names used to insert logical control characters in line mode output messages or to detect line mode inputs

5 Description of the macros

This section contains detailed descriptions of the individual Executive macros in alphabetical order. In general the descriptions are arranged into the following parts:

– macro name and function
– general: application area and macro type
– macro description: macro function
– macro format and operand description
– return information and error flags: return code and explanation
– example

The TIAM macros described in this manual reflect the functional scope of the current TIAM version, V13.2A (see also the “TIAM” manual [16]):

CUPAB **RDATA** **TCHNG** **TSTAT**

WROUT **WRTRD**

The VTSU macros described in this manual, reflect the functional scope of the current VTSU V13.3A (see the “VTSU” manual [30]):

DCSTA **VTCSET** **VTSUCB**

The DBL macros described in this manual, reflect the functional scope of the current binder loader BLSSERV V2.8A ; (see the “BLSSERV” manual [4]):

ASHARE **BIND** **DSHARE** **ETABIT** **ETABLE**

GETPRGV **ILEMGT** **ILEMIT** **LDSLICE** **PINF**

SELPRGV **UNBIND** **VSVI1**

AINF – Measure resource utilization

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Type S, MF format **1**: 31-bit interface: standard/E/L/D form; see page 29

Resources whose utilization is to be measured are defined in so-called information packages. The following information packages can be selected for measuring:

Designation	Resources/Measured values
Global values	CPU time, total number of inputs and outputs, number of data blocks transferred, and working set integral are determined.
Time usage	CPU time, time stamp or run time are determined.
Input/output count	Number of inputs and outputs.

A description of the structure of the output fields for the individual information packages follows after the description of the macro format.

Working set integral: Sum of all products of (main memory pages in Kbytes * user time in seconds).
Number of data blocks transferred: transport of data to and from the local periphery, in PAM pages (for public, system private and user private disks) or in 2K blocks (for magnetic tapes and unit record devices).

Macro description

The **AINF** macro determines the resource utilization of a job and passes the values to a user program area.
Two methods are available for measuring resource utilization:

– Usage stamp method:
  Resource utilization since the beginning of the job (format 1)

– Measurement method:
  Resource utilization of individual program sections (formats 2 and 3)

The MF=D operand generates a DSECT for the data area and a definition of possible return information items. Additionally, the **AINF** macro may output the definitions of the output structure of individual information packages (format 4).

Note

> The CPU time consumed by the system call processing itself corrupts the measurement slightly.

Functional description

When the **usage stamp method** is used, the **AINF** macro determines resource usage since the beginning of the job and transfers the usage values to a user program area. The user selects the resource to be measured by specifying information packages in the macro (GLOBAL, TIME, IOCNT) to which different resources are assigned.

When the **measurement method** is used, the **AINF** macro determines the resource usage of individual program sections. Measurement may be started, interrupted, continued or terminated at freely selectable points in the program. Several measurements may be nested and overlap as required. Each measurement is assigned its own measurement ID in order to allow unique identification of different measurements.
When a measurement is started (READY operand), the user defines a measurement ID and selects the information packages containing the resources to be measured. If the measurement is to be interrupted, the user must issue another **AINF** call (with the INTR operand) at the point where the interrupt is to take place. The system determines the usage values of the specified resources since the start of the measurement and transfers them, if desired, to a user program area. To continue the interrupted measurement, the user must issue an **AINF** call (with the READY operand) containing the associated measurement ID at the desired point. In this way, a measurement may be interrupted whenever desired and continued later on. Following each interrupt, the system determines the total of the usage values of all measurement steps up to this point.
Measurement is terminated when an **AINF** call (with the FINISH operand) is issued or the program is terminated. The system transfers the total of the usage values of all measuring steps (from the start of measurement to the end) to a user program area.

AINF data area format:

Field name	Displacement	Contents
`IAMID`	`00`	Macro ID
`IAIMFC`	`04`	Function selection
`IAIMTARE`	`08`	Operand type
`IAIMAREA`	`0C`	Output area
`IAIMCHAI`	`10`	Chaining address
`IAIMMID`	`14`	Measurement ID

Macro formats and description of operands

The operands of the following descriptions are described in alphabetical order.

Format 1: Usage stamp method

AINF
AREA=addr / (r) ,GLOBAL=**NO** / **Y**ES ,TIME=**NO** / **Y**ES ,IOCNT=**NO** / **Y**ES / EXT / STD ,MF=S / (E,..) / L

At least one of the operands GLOBAL, TIME or IOCNT must be specified with the value YES, otherwise the macro is rejected with:

- X'10' in register R15 if only default values (NO) were specified explicitly.
- an MNOTE message (in the ASSEMBLER log) if none of the operands GLOBAL, TIME or IOCNT were specified explicitly.

AREA=
Specifies a field where resource utilization values are entered. The field must be aligned on a word boundary. The length of the field is determined by the desired information package (see “Output structure format” on page 178).

addr
Symbolic address of the field.

(r)
Register containing the address value “addr”.

GLOBAL=
Specifies the information package “global values”.

NO
The information package “global values” is not selected.

YES
Resource utilization is output.

IOCNT=
Specifies the information package “input/output count”.

NO
The information package “input/output count” is not selected.

YES
The total number of input/output operations to/from public volumes and system private disks is output.

STD
The total number of input/output operations to/from public volumes and system private disks is output.

EXT
Same as STD or YES, but includes input/output operations to/from tapes, user private disks and other devices.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

TIME=
Specifies the information package “time usage”.

NO
The information package “time usage” is not selected.

YES
CPU time and time stamp (time of day) are to be measured. The time stamp in the usage stamp method is the value of the TOD register converted to seconds and nanoseconds.

Format 2: Measurement method
Start or restart measurement

AINF
READY='measid' / addr / (r) ,GLOBAL=**NO** / **Y**ES ,TIME=**NO** / **Y**ES ,IOCNT=**NO** / **Y**ES / EXT / STD [,CHAIN=addr / (r)] ,MF=S / (E,..) / L

At least one of the operands GLOBAL, TIME or IOCNT must be specified with the value YES, otherwise the macro is rejected with:

- X'10' in register R15 if only default values (NO) were specified explicitly.
- an MNOTE message (in the ASSEMBLER log) if none of the operands GLOBAL, TIME or IOCNT were specified explicitly.

CHAIN=
Permits the **AINF** macro to be chained to another **AINF** macro by specifying a chaining address. This address refers to the data area of the second **AINF** macro generated with MF=L. See notes on chaining (page 181).

addr
Address of the operand list of the second macro generated with MF=L.

(r)
Register containing the address value “addr”. This operand is permitted only for the standard form or the L form of the macro.

GLOBAL=
Specifies the information package “global values”.

NO
The information package “global values” is not selected.

YES
Resource utilization is output.

IOCNT=
Specifies the information package “input/output count”.

NO
The information package “input/output count” is not selected.

YES
The total number of input/output operations to/from public volumes and system private disks is output.

STD
The total number of input/output operations to/from public volumes and system private disks is output.

EXT
Same as STD or YES, but includes input/output operations to/from tapes, user private disks and other devices.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

READY=
Starts a new measurement and assigns it the specified measurement ID.

'measid'
Measurement ID assigned to the measurement. Length: up to 8 characters.
If the 'measid' name is already assigned to an (interrupted) measurement, this measurement is restarted.

addr
Symbolic name (address) of a field containing the measurement ID.

(r)
Register containing the address value “addr”.

TIME=
Specifies the information package “time usage”.

<u>NO</u>
The information package “time usage” is not selected.

YES
CPU time and runtime are to be output.

Note

When a measurement is restarted, the selection of measurement packages (GLOBAL, TIME, IOCNT) must match the specifications made when the measurement was originally started. If this is not the case, the measurement is continued with the information packages originally specified and the return information X'24' is stored in register R15.

Format 3: Measurement method
Interrupt or terminate measurement

AINF

{ INTR='measid' / addr / (r)[,AREA=addr / (r)] }
{ FINISH='measid' / addr / (r),AREA=addr / (r) }
[,CHAIN=addr / (r)]
,MF=S / (E,..) / L

AREA=
Specifies a field where resource utilization values are entered. The field must be aligned on a word boundary. The length of the field is determined by the desired information package (see “Output structure format” on page 178).

addr
Symbolic address of the field.

(r)
Register containing the address value “addr”.

CHAIN=
Permits the **AINF** macro to be chained to another **AINF** macro by specifying a chaining address. This address refers to the operand list of the second **AINF** macro generated with MF=L. See notes on chaining (page 181).

addr
Address of the operand list of the second macro generated with MF=L.

(r)
Register containing the address value “addr”. This operand is permitted only for the standard form or the L form of the macro.

FINISH=
FINISH terminates the measurement with the specified measurement ID.

'measid'
Measurement ID assigned to the measurement.

addr
Symbolic address of a field containing the measurement ID.

(r)
Register containing the address value “addr”.

INTR=
INTR interrupts the measurement with the specified measurement ID.

'measid'
Measurement ID assigned to the measurement.

addr
Symbolic address of a field containing the measurement ID.

(r)
Register containing the address value “addr”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

Format 4: DSECTs for operand list and output structure
(at least one operand must be specified)

AINF
[MF=D] ,P=I / P / * [,GLOBAL=D] [,TIME=D] [,IOCNT=D]

GLOBAL=D
Generates a DSECT for the output structure of the information package “global values”.

IOCNT=D
Generates a DSECT for the output structure of the information package “input/output count”.

MF=D
Generates a DSECT for the operand list and EQU statements for the return code.

P=
Specifies a prefix for the symbolic names of the DSECT.

> **I**
> The generated field names begin with the prefix `I`, the length definitions with `LIxx`.
>
> **p**
> Letter prefixed to all generated field names of the DSECT. The prefix is also included in the names of the length definitions: `L&p.xx`.
> Length of the prefix = 1 letter.
>
> *****
> No letter is prefixed to the generated field names or included in the names of the length definitions.

TIME=D
Generates a DSECT for the output structure of the information package “time usage”.

```
           AINF  MF=D
1          #INTF INTNAME=AINF,REFTYPE=REQUEST,INTCOMP=001          GS 950
1          MFPRE DNAME=AIMPL,MF=D,PREFIX=I,MACID=AIM,DMACID=AIM    :*R200
2 IAIMPL   DSECT ,
2                *,##### PREFIX=I, MACID=AIM #####
1          DS    0F                      AINF PARAMETER LIST
1 IAIMID   DS    CL2                     IDENTIFICATION OF AINF MACRO
1 IAIMAINF EQU   C'AI'                   - AINF INFORMATION STAMP
1 IAIMPMAC EQU   C'PM'                   - PROG MEASUREMENT FUNCT.
1 IAIMVER  DC    XL2'075'                 AINF MACRO VERSION
1 IAIMFC   DS    X                       FUNCTION CODE
1 IAIMFCCV EQU   X'01'                   - CURRENT USAGE VALUES
1 IAIMFCRE EQU   X'02'                   - MEASUREMENT READY
1 IAIMFCIN EQU   X'04'                   - MEASUREMENT INTR
1 IAIMFCFI EQU   X'08'                   - MEASUREMENT FINISH
1 IAIMNPAR DC    FL1'4'                     # OF PARAMETERS
1 *                                        + 8 = DIST OF FIRST PARAM ADDR
1 IAIMMSK1 DC    X'00'                   INFORMATION PACKAGE MASK1
1 IAIMGLOB EQU   X'80'                   - "GLOBAL"    REQUIRED
1 IAIMTIME EQU   X'40'                   - "TIME"      REQUIRED
1 IAIMIOCN EQU   X'20'                   - "IOCNT=STD" REQUIRED
1 IAIMIOCX EQU   X'10'                   - "IOCNT=EXT" REQUIRED
1 *        EQU   X'0F'                   - ALL OTHER BITS RESERVED
1 IAIMMSK2 DC    X'00'                   INFORMATION PACKAGE MASK2
1 *        EQU   X'FF'                   - ALL BITS RESERVED
1 IAIMTARE DS    X                       TYPE OF AREA PARAM
1 IAIMNONE EQU   X'00'                   - NOT SPECIFIED
1 IAIMADDR EQU   X'01'                   - GIVEN AS DIRECT ADDRESS
1 IAIMREG  EQU   X'02'                   - GIVEN IN A REGISTER
1 IAIMTCHA DS    X                       TYPE OF CHAIN PARAM
1 *                                      - EQUATES AS ABOVE
1 IAIMTMID DS    X                       TYPE OF MEASUREMENT ID
1 *                                      - EQUATES AS ABOVE, +
1 IAIMSTR  EQU   X'03'                   - GIVEN AS A STRING
1 IAIMTMIX DS    X                       TYPE OF MID CONTINUED
1 IAIMEXT  EQU   X'FF'                   - EXTENDED PARAMETER
1 IAIMAREA DS    A                       OUTPUT AREA ADDRESS
1 IAIMCHAI DS    A                       CHAIN ADDRESS
1 IAIMMID  DS    0A                      MEASUREMENT ID ADDR  OR
1 IAIMMIDS DS    CL8                     MEASUREMENT ID STRING
1 IAIMPLE  EQU   *                       END OF AINF PARAM LIST
1 LIAIMPL  EQU   *-IAIMPL               LENGTH OF AINF PARAM LIST
1          SPACE 2
```

The DSECT is followed by the definitions of possible return codes. This part of the expansion is listed below under “Return information and error flags”.

Output structure format

1. Information package **Global values**

Field name	Displacement	Contents
IAIGTCPU	00	CPU time in the following format: Displacement 00: full seconds Displacement 04: remainder in nanoseconds Current measurement accuracy: 1 microsecond
IAIG#IOS	08	Total number of inputs/outputs
	0C	Number of data blocks
	10	Working set integral

```
          AINF  GLOBAL=D
1         #INTF INTNAME=AINF,REFTYPE=REQUEST,INTCOMP=001          GS 950
1         MFPRE DNAME=AIAREA,MF=D,PREFIX=I,MACID=AIA,DMACID=AIA   :*R200
2 IAIAREA  DSECT ,
2               *,##### PREFIX=I, MACID=AIA #####
1         DS    0F                       START OF OUTPUT AREA
1         MFPRE DNAME=AIGLOB,ALIGN=F,PREFIX=I,MACID=AIG,MF=S,     :*R200C
1               DMACID=AIG                                        :*R200
2         CNOP  0,4
2 IAIGLOB  DS    0F
1 *                                      "GLOBAL" INFO PACKAGE
1 IAIGTCPU DS    0FL8                    CPU TIME
1 IAIGCPUS DS    F                       CPU TIME SECONDS
1 IAIGCPUN DS    F                       CPU TIME NANOSECONDS
1 IAIG#IOS DS    F                       TOTAL # IO’S
1 IAIG#BLK DS    F                       TOTAL # BLOCKS           GS 090
1 IAIGWSI  DS    FL8                     WORKING SET INTEGRAL     GS 090
1 IAIGLOBE EQU   *                       END OF "GLOBAL" INFO
1 LIAIGLOB EQU   *-IAIGLOB              LENGTH OF "GLOBAL" INFO
1         SPACE 2
1 *                                      END OF OUTPUT AREA
1         MFPRE DNAME=AIAEND,ALIGN=F,PREFIX=I,MACID=AIA,MF=S,     :*R200C
1               DMACID=AIA                                        :*R200
2         CNOP  0,4
2 IAIAEND  DS    0F
1 LIAIAREA EQU   *-IAIAREA              LENGTH OF OUTPUT AREA
1         SPACE 2
```

The measured CPU time is composed of:

- CPU time in nonprivileged program mode
- CPU time in privileged program mode (SVC call and program error processing)
- CPU time during command processing

2. Information package **Time usage**

Field name	Displacement	Contents
IAITTCPU	00	CPU time in the following format: Displacement 00: full seconds Displacement 04: remainder in nanoseconds Current measurement accuracy: 1 microsecond
IAITETIM	08	For usage stamp method: time stamp For measurement method: runtime Displacement 08: full seconds Displacement 0C: remainder in nanoseconds Current measuring accuracy: 1 microsecond

```
          AINF  TIME=D
1         #INTF INTNAME=AINF,REFTYPE=REQUEST,INTCOMP=001          GS 950
1         MFPRE DNAME=AIAREA,MF=D,PREFIX=I,MACID=AIA,DMACID=AIA   :*R200
2 IAIAREA DSECT ,
2               *,##### PREFIX=I, MACID=AIA #####
1         DS    0F                       START OF OUTPUT AREA
1         MFPRE DNAME=AITIME,ALIGN=F,PREFIX=I,MACID=AIT,MF=S,     :*R200C
1               DMACID=AIT                                        :*R200
2         CNOP  0,4
2 IAITIME DS    0F
1 *                                      "TIME"   INFO PACKAGE
1 IAITTCPU DS   0FL8                     CPU TIME
1 IAITCPUS DS   F                        CPU TIME SECONDS
1 IAITCPUN DS   F                        CPU TIME NANOSECONDS
1 IAITETIM DS   0FL8                     ELAPSED TIME / TIME STAMP
1 IAITTIMS DS   F                        TIME IN SECONDS
1 IAITTIMN DS   F                        TIME NANOSECONDS
1 IAITIMEE EQU  *                        END OF "TIME"   INFO
1 LIAITIME EQU  *-IAITIME               LENGTH OF "TIME"   INFO
1         SPACE 2
1 *                                      END OF OUTPUT AREA
1         MFPRE DNAME=AIAEND,ALIGN=F,PREFIX=I,MACID=AIA,MF=S,     :*R200C
1               DMACID=AIA                                        :*R200
2         CNOP  0,4
2 IAIAEND DS    0F
1 LIAIAREA EQU  *-IAIAREA               LENGTH OF OUTPUT AREA
1         SPACE 2
```

A breakdown of CPU time is given in the “global values” information package.
The time stamp in the usage stamp method is the value of the TOD register converted into seconds and nanoseconds.

3. Information package **Input/output count**

Field name	Displacement	Contents
IAII#IOS	00	Input/output total
IAIIIOPD	04	Input/output on public volumes
IAIIIOSD	08	Input/output on shareable private disks
IAIIIOUD	0C	Input/output on task-exclusive disks
IAIIIOTP	10	Input/output on tape
IAIIIOUR	14	Input/output on other devices

```
           AINF  IOCNT=D
1          #INTF INTNAME=AINF,REFTYPE=REQUEST,INTCOMP=001          GS 950
1          MFPRE DNAME=AIAREA,MF=D,PREFIX=I,MACID=AIA,DMACID=AIA   :*R200
2 IAIAREA  DSECT ,
2                *,##### PREFIX=I, MACID=AIA #####
1          DS    0F                      START OF OUTPUT AREA
1          MFPRE DNAME=AIIOCN,ALIGN=F,PREFIX=I,MACID=AII,MF=S :*R200DMACIC
1                D=AII                                        :*R200
2          CNOP  0,4
2 IAIIOCN  DS    0F
1 *                                      "IOCNT"  INFO PACKAGE
1 IAII#IOS DS    F                       TOTAL # IO’S
1 IAIIIOPD DS    F                # IO’S ON PUBLIC DEVICES
1 IAIIIOSD DS    F                # IO’S ON SYSTEM PRIV. DISKS
1 IAIIIOUD DS    F                # IO’S ON USER PRIVATE DISKS
1 IAIIIOTP DS    F                # IO’S ON TAPE DEVICES
1 IAIIIOUR DS    F                # IO’S ON UNIT RECORD DEVICES
1 IAIIOCNE EQU   *                       END OF "IOCNT"  INFO
1 LIAIIOCN EQU   *-IAIIOCN              LENGTH OF "IOCNT"  INFO
1          SPACE 2
1 *                                      END OF OUTPUT AREA
1          MFPRE DNAME=AIAEND,ALIGN=F,PREFIX=I,MACID=AIA,MF=S,     C
1                DMACID=AIA
2          CNOP  0,4
2 IAIAEND  DS    0F
1 LIAIAREA EQU   *-IAIAREA              LENGTH OF OUTPUT AREA
```

The following are not included in the input/output count:

- Terminal inputs/outputs for interactive and transaction processes
- Paging inputs/outputs

The following applies in addition to single inputs/outputs to tapes, user private disks and other devices (see the IOCNT=EXT operand):
The following are not included in the input/output count

- I/Os from privileged programs
- I/Os to devices released prior to the program run (by REMOVE-FILE-LINK or SECURE-RESOURCE-ALLOCATION)

As these restrictions do not apply to the input/output totals, the value computed for the total number may be greater than the sum of single values.

Notes on chaining for measurement method (CHAIN operand)

– The **AINF** data area designated by CHAIN is processed immediately after the data area of the calling **AINF** macro. This allows several **AINF** calls to be executed within one SVC processing session; e.g. terminating or interrupting a measurement with concurrent start or restart of a new measurement (under a different measurement ID); or interrupting a measurement to output current measuring results with immediate restart of the same measurement.

– Address transfer in registers facilitates reentrant programming but is not recommended unless no more than two operands are to be linked.

– Following SVC processing, register R1 contains the address of the most recently processed data area.

– If an error occurs in the **AINF** data area of a chain, the chain is terminated at this point. Register R1 then contains the address of the errored data area.

Return information and error flags

Register R1 contains the address of the data area; for macro chaining, it contains the address of the most recently processed data area..

R15:

						a	a

A return code relating to the execution of the AINF macro is transferred in the rightmost byte of register R15; the remaining three bytes are deleted.

Field name	X'aa'	Meaning
IAIROK	X'00'	Call executed successfully.
IAIRADER	X'04'	Call not executed due to invalid operand list address.
IAIRFUER	X'08'	Call not executed due to invalid macro ID or invalid function code (neither usage stamp method nor READY, INTR or FINISH).
IAIRCHER	X'0C'	Call executed despite invalid chaining address in most recently processed operand block (READY, INTR, FINISH).
IAIRPAER	X'10'	Call not executed due to: – invalid register or output area address; – invalid register or measurement ID address; – missing operand entries.
IAIRMRDY	X'14'	Call not executed because measurement is already being performed under the specified measurement ID (with READY).
IAIRNCL5	X'18'	Call not executed because there is no more class 5 memory available for work area (with READY).
IAIRMNTF	X'1C'	Call not executed because INTR or FINISH had no associated READY; i.e. measurement not yet started.
IAIRMINT	X'20'	Call executed despite FINISH for a measurement interrupted by means of INTR; i.e. measurement not yet resumed.
IAIRIRIN	X'24'	Call executed despite fact that READY call for measurement resumption (after INTR) does not contain same information as READY call when measurement was started.

During macro chaining, the specification “Call executed / not executed?” refers to the most recently processed data area (i.e. the data area whose address is stored in register R1). If an error occurs during chaining, the chain is terminated at the point of the errored data area, where:

“executed” indicates that the last (errored) data area was still processed. If the error can be corrected by the user program, continuation of processing starting at the next data area should be requested.

“not executed” indicates that the last (errored) data area could not be processed. If the error can be corrected by the user program, processing should continue with the same (corrected) operand list.

Example 1: Usage stamp method

```
AINF1    START
         PRINT NOGEN
         BALR  3,0
         USING *,3
         LA    5,AREAONE
         USING AINFDATA,5
         AINF  GLOBAL=Y,TIME=Y,IOCNT=EXT,AREA=AREAONE ───────────────  (1)
         PRINT NOGEN
*
*        Display CPU time
*
         MVC   MESSTXT(L'MCPU),MCPU
         L     8,IAIGCPUS             CPU time in seconds
         BAL   7,PKD2ZND              Call conversion routine ------>
         MVC   MESSTXT+20(10),ASSIST2+6
         MVI   MESSTXT+30,C'.'
         L     8,IAIGCPUN             CPU time in nanoseconds
         BAL   7,PKD2ZND              Call conversion routine ------>
         MVC   MESSTXT+31(9),ASSIST2+7
         BAL   7,OUTPUT               Call output routine ---------->
*
*        Display elapsed time
*
         MVC   MESSTXT(L'MELA),MELA
         L     8,IAITTIMS             Elapsed time in seconds
         BAL   7,PKD2ZND              Call conversion routine ------>
         MVC   MESSTXT+20(10),ASSIST2+6
         MVI   MESSTXT+30,C'.'
         L     8,IAITTIMN             Elapsed time in nanoseconds
         BAL   7,PKD2ZND              Call conversion routine ------>
         MVC   MESSTXT+31(9),ASSIST2+7
         BAL   7,OUTPUT               Call output routine ---------->
*
*        Display # IO's
*
         MVC   MESSTXT(L'MIOS),MIOS   * Total # IO's ****************
         L     8,IAII#IOS
         BAL   7,PKD2ZND              Call conversion routine ------>
         MVC   MESSTXT+30(10),ASSIST2+6
         BAL   7,OUTPUT               Call output routine ---------->
*
         MVC   MESSTXT(L'MIOP),MIOP   * Public IO's *****************
         L     8,IAIIIOPD
         BAL   7,PKD2ZND              Call conversion routine ------>
         MVC   MESSTXT+30(10),ASSIST2+6
         BAL   7,OUTPUT               Call output routine ---------->
```

```
  *
           MVC   MESSTXT(L'MIOSP),MIOSP  * System private IO's *********
           L     8,IAIIIOSD
           BAL   7,PKD2ZND               Call conversion routine ------>
           MVC   MESSTXT+30(10),ASSIST2+6
           BAL   7,OUTPUT                Call output routine ---------->
  *
           MVC   MESSTXT(L'MIOSP),MIOSP  * User private IO's ***********
           L     8,IAIIIOUD

           BAL   7,PKD2ZND               Call conversion routine ------>
           MVC   MESSTXT+30(10),ASSIST2+6
           BAL   7,OUTPUT                Call output routine ---------->
  *
           MVC   MESSTXT(L'MITP),MITP    * Tape devices ****************
           L     8,IAIIIOTP
           BAL   7,PKD2ZND               Call conversion routine ------>
           MVC   MESSTXT+30(10),ASSIST2+6
           BAL   7,OUTPUT                Call output routine ---------->
  *
           MVC   MESSTXT(L'MIUR),MIUR    * Unit record devices *********
           L     8,IAIIIOUR
           BAL   7,PKD2ZND               Call conversion routine ------>
           MVC   MESSTXT+30(10),ASSIST2+6
           BAL   7,OUTPUT                Call output routine ---------->
  END      TERM
  *
  *        Output routine
  *
  OUTPUT   WROUT MESSAGE,END,PARMOD=31
2                *,@DCEO      999    921011   53531004
           MVI   MESSTXT,C' '            Clear MESSTXT for next output
           MVC   MESSTXT+1(L'MESSTXT-1),MESSTXT
           BR    7                       Return ->
  *
  *        Conversion routine
  *
  PKD2ZND  CVD   8,ASSIST1               Convert register contents to
           UNPK  ASSIST2,ASSIST1         zoned decimal
           MVZ   ASSIST2+15(1),=X'F0'
           BR    7                       Return ->
  *
  *        Definitions
  *
  AREAONE  DS    0F
           DS    6F
           DS    4F
           DS    6F
  MESSAGE  DC    Y(ENDMESS-MESSAGE)      Record length
           DS    CL2                     Reserved
           DC    X'01'                   Print feed control character
  MESSTXT  DC    CL60' '                 Text
  ENDMESS  EQU   *
  ASSIST1  DS    D
  ASSIST2  DS    CL16
  MCPU     DC    C'CPU Time used:'
  MELA     DC    C'Elapsed Time used:'
  MIOS     DC    C'Total number of IOs:'
  MIOP     DC    C'IOs on public devices:'
  MIOSP    DC    C'IOs on system private disks:'
  MIOUP    DC    C'IOs on user private disks:'
  MITP     DC    C'IOs on tape devices:'
  MIUR     DC    C'IOs on unit record devices:'
           PRINT GEN
  AINFDATA AINF  GLOBAL=D,TIME=D,IOCNT=D ——————————————————————————————  (2)
1          #INTF INTNAME=AINF,REFTYPE=REQUEST,INTCOMP=001          GS 950
1 AINFDATA MFPRE DNAME=AIAREA,MF=D,PREFIX=I,MACID=AIA,DMACID=AIA   :*R200
2 AINFDATA DSECT ,
2                *,##### PREFIX=I, MACID=AIA #####
1 IAIAREA  DS    0F                      START OF OUTPUT AREA      LKH075
1          MFPRE DNAME=AIGLOB,ALIGN=F,PREFIX=I,MACID=AIG,MF=S,     :*R200C
1                DMACID=AIG                                        :*R200
2          CNOP  0,4
2 IAIGLOB  DS    0F
1 *                                      "GLOBAL" INFO PACKAGE     LKH075
1 IAIGTCPU DS    0FL8                    CPU TIME                  LKH075
1 IAIGCPUS DS    F                       CPU TIME SECONDS          LKH075
1 IAIGCPUN DS    F                       CPU TIME NANOSECONDS      LKH075
1 IAIG#IOS DS    F                       TOTAL # IO'S              LKH075
1 IAIG#BLK DS    F                       TOTAL # BLOCKS            GS 090
1 IAIGWSI  DS    FL8                     WORKING SET INTEGRAL      GS 090
1 IAIGLOBE EQU   *                       END OF "GLOBAL" INFO      LKH075
1 LIAIGLOB EQU   *-IAIGLOB              LENGTH OF "GLOBAL" INFO   LKH075
1          SPACE 2                                                 LKH075
1          MFPRE DNAME=AITIME,ALIGN=F,PREFIX=I,MACID=AIT,MF=S,     :*R200C
1                DMACID=AIT                                        :*R200
2          CNOP  0,4
2 IAITIME  DS    0F
1 *                                      "TIME"   INFO PACKAGE     LKH075
1 IAITTCPU DS    0FL8                    CPU TIME                  LKH075
1 IAITCPUS DS    F                       CPU TIME SECONDS          LKH075
1 IAITCPUN DS    F                       CPU TIME NANOSECONDS      LKH075
1 IAITETIM DS    0FL8                    ELAPSED TIME / TIME STAMP LKH075
1 IAITTIMS DS    F                       TIME IN SECONDS           LKH075
1 IAITTIMN DS    F                       TIME NANOSECONDS          LKH075
1 IAITIMEE EQU   *                       END OF "TIME"   INFO      LKH075
```

```
1 LIAITIME EQU   *-IAITIME              LENGTH OF "TIME"   INFO   LKH075
1          SPACE 2                                                 LKH075
1          MFPRE DNAME=AIIOCN,ALIGN=F,PREFIX=I,MACID=AII,MF=S :*R200DMACIC
1                D=AII                                        :*R200
2          CNOP  0,4
2 IAIIOCN  DS    OF
1 *                                      "IOCNT"  INFO PACKAGE     LKH075
1 IAII#IOS DS    F                       TOTAL # IO'S              LKH075
1 IAIIIOPD DS    F                # IO'S ON PUBLIC DEVICES         LKH075
1 IAIIIOSD DS    F                # IO'S ON SYSTEM PRIV. DISKS     LKH075
1 IAIIIOUD DS    F                # IO'S ON USER PRIVATE DISKS     LKH075
1 IAIIIOTP DS    F                # IO'S ON TAPE DEVICES           LKH075
1 IAIIIOUR DS    F                # IO'S ON UNIT RECORD DEVICES    LKH075
1 IAIIOCNE EQU   *                       END OF "IOCNT"  INFO      LKH075
1 LIAIIOCN EQU   *-IAIIOCN              LENGTH OF "IOCNT"  INFO   LKH075
1          SPACE 2                                                 LKH075
1 *                                      END OF OUTPUT AREA        LKH075
1          MFPRE DNAME=AIAEND,ALIGN=F,PREFIX=I,MACID=AIA,MF=S,     :*R200C
1                DMACID=AIA                                        :*R200
2          CNOP  0,4
2 IAIAEND  DS    OF
1 LIAIAREA EQU   *-IAIAREA              LENGTH OF OUTPUT AREA     LKH075
1          SPACE 2                                                 LKH075
           END
                 =X'F0'
```

(1) Resource utilization since the start of the job is to be measured using the information packages “global values”, “time usage” and “input/output count”. The measurement values are to be transferred to the area AREAONE.

(2) A DSECT is generated for each of the different information packages. The field names defined there may be used for symbolic addressing of the measurement values in AREAONE.

Runtime log

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,ainf1), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,ainf1))
%  ASS6011 ASSEMBLY TIME: 380 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 107 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=ainf1
%  BLS0523 ELEMENT 'AINF1', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'AINF1', VERSION ' ' OF '<date> <time>' LOADED
CPU Time used:      0000000058.526666000 ------------------------------  (3)
Elapsed Time used:  0000001333.942669000
Total number of IOs:          0000019462
IOs on public devices:        0000019462
IOs on system private disks:  0000000000
IOs on system private disks:  0000000000
IOs on tape devices:          0000000000
IOs on unit record devices:   0000000000
```

(3) The calculated measurement values are output.

Example 2: Measurement method

In the AINF2 program, two measurements are started. These measurements are assigned the codes MES 1 and MES 2.
MES1 requests resource utilisation data which corresponds to the information packages “time usage” and “input and output tally”. When the measurement is terminated, the information should be output to AREA1.
MES2 requests resource utilisation data which corresponds to the information packages “global values” and “input and output tally”. When the measurement is suspended, i.e. at the end of the first measurement period, the information should be output to AREA2A; when the measurement is terminated, to AREA2B.
In addition, the DSECTS for the output structures of the information packages are generated for MES1 and MES2 and are linked to the relevant output areas.

```
AINF2    START
         PRINT NOGEN
         BALR  3,0
         USING *,3
MES1B    AINF  READY='MES1',TIME=Y,IOCNT=EXT    Start MES1
         LTR   15,15                  If Returncode not zero
         BNE   ERROR                  go to ERROR ->
         MVC   MESSTXT,STARTMSG
         BAL   7,OUTPUT               Call output routine ---------->
MES2B    AINF  READY='MES2',GLOBAL=Y,IOCNT=Y             Start MES2
         LTR   15,15                  If Returncode not zero
         BNE   ERROR                  go to ERROR ->
         MVI   STARTMSG+3,C'2'        Modify start message
         MVC   MESSTXT,STARTMSG
         BAL   7,OUTPUT               Call output routine ---------->
MES2I    AINF  INTR='MES2',AREA=AREA2A                   Interrupt MES2
MES2C    AINF  READY='MES2',GLOBAL=Y,IOCNT=Y             Continue MES2
         BKPT
MES2E    AINF  FINISH='MES2',AREA=AREA2B                 Terminate MES2
MES1E    AINF  FINISH='MES1',AREA=AREA1         Terminate MES1
*
*        Display Elapsed Time (MES1)
*
         USING MES1PAR,5
         LA    5,AREA1
         MVC   MESSTXT(L'MELA),MELA
         L     8,AAITTIMS             Elapsed time in seconds
         BAL   7,PKD2ZND              Call conversion routine ------>
         MVC   MESSTXT+20(10),ASSIST2+6
         MVI   MESSTXT+30,C'.'
         L     8,AAITTIMN             Elapsed time in nanoseconds
         BAL   7,PKD2ZND              Call conversion routine
         MVC   MESSTXT+31(9),ASSIST2+7
```

```
           BAL   7,OUTPUT                Call output routine ---------->
  *
  *        Display # IO's at interrupt (MES2)
  *
           USING MES2PAR,6
           LA    6,AREA2A
           MVC   MESSTXT(L'MIOPI),MIOPI  * Public IO's *****************
           L     8,BAIIIOPD
           BAL   7,PKD2ZND               Call conversion routine ------>
           MVC   MESSTXT+30(10),ASSIST2+6
           BAL   7,OUTPUT                Call output routine ---------->
  *
  *        Display # IO's at end (MES2)
  *
           LA    6,AREA2B
           MVC   MESSTXT(L'MIOPE),MIOPE  * Public IO's *****************
           L     8,BAIIIOPD
           BAL   7,PKD2ZND               Call conversion routine ------>
           MVC   MESSTXT+30(10),ASSIST2+6
           BAL   7,OUTPUT                Call output routine ---------->
  END      TERM
  ERROR    NOP   ERROR

  *        :                Error handling
           TERM
  *
  *        Output routine
  *
  OUTPUT   WROUT MESSAGE,END,PARMOD=31
2                *,@DCEO      999    921011   53531004
           MVI   MESSTXT,C' '            Clear MESSTXT for next output
           MVC   MESSTXT+1(L'MESSTXT-1),MESSTXT
           BR    7                       Return ->
  *
  *        Conversion routine
  *
  PKD2ZND  CVD   8,ASSIST1               Convert register contents to
           UNPK  ASSIST2,ASSIST1         zoned decimal
           MVZ   ASSIST2+15(1),=X'F0'
           BR    7                       Return ->
  AREA1    DS    10F
  AREA2A   DS    12F
  AREA2B   DS    12F
  MESSAGE  DC    Y(ENDMESS-MESSAGE)      Record length
           DS    CL2                     Reserved
           DC    X'01'                   Print feed control character
  MESSTXT  DC    CL60' '                 Text
  ENDMESS  EQU   *
  STARTMSG DC    CL60'MES1 successfully started'
  ASSIST1  DS    D
  ASSIST2  DS    CL16
  MCPU     DC    C'CPU Time used:'
  MELA     DC    C'Elapsed Time used:'
  MIOPI    DC    C'IOs before BKPT:'
  MIOPE    DC    C'IOs after BKPT:'
           PRINT GEN
  MES1PAR  AINF  TIME=D,IOCNT=D,P=A
1          #INTF INTNAME=AINF,REFTYPE=REQUEST,INTCOMP=001          GS 950
1 MES1PAR  MFPRE DNAME=AIAREA,MF=D,PREFIX=A,MACID=AIA,DMACID=AIA   :*R200
2 MES1PAR  DSECT ,
2                *,##### PREFIX=A, MACID=AIA #####
1 AAIAREA  DS    0F                       START OF OUTPUT AREA      LKH075
1          MFPRE DNAME=AITIME,ALIGN=F,PREFIX=A,MACID=AIT,MF=S,      :*R200C
1                DMACID=AIT                                         :*R200
2          CNOP  0,4
2 AAITIME  DS    0F
1 *                                       "TIME"   INFO PACKAGE     LKH075
1 AAITTCPU DS    0FL8                     CPU TIME                  LKH075
1 AAITCPUS DS    F                        CPU TIME SECONDS          LKH075
1 AAITCPUN DS    F                        CPU TIME NANOSECONDS      LKH075
1 AAITETIM DS    0FL8                     ELAPSED TIME / TIME STAMP LKH075
1 AAITTIMS DS    F                        TIME IN SECONDS           LKH075
1 AAITTIMN DS    F                        TIME NANOSECONDS          LKH075
1 AAITIMEE EQU   *                        END OF "TIME"   INFO      LKH075
1 LAAITIME EQU   *-AAITIME               LENGTH OF "TIME"   INFO   LKH075
1          SPACE 2                                                  LKH075

1          MFPRE DNAME=AIIOCN,ALIGN=F,PREFIX=A,MACID=AII,MF=S :*R200DMACIC
1                D=AII                                         :*R200
2          CNOP  0,4
2 AAIIOCN  DS    0F
1 *                                       "IOCNT"  INFO PACKAGE     LKH075
1 AAII#IOS DS    F                        TOTAL # IO'S              LKH075
1 AAIIIOPD DS    F                 # IO'S ON PUBLIC DEVICES         LKH075
1 AAIIIOSD DS    F                 # IO'S ON SYSTEM PRIV. DISKS     LKH075
1 AAIIIOUD DS    F                 # IO'S ON USER PRIVATE DISKS     LKH075
1 AAIIIOTP DS    F                 # IO'S ON TAPE DEVICES           LKH075
1 AAIIIOUR DS    F                 # IO'S ON UNIT RECORD DEVICES    LKH075
1 AAIIOCNE EQU   *                        END OF "IOCNT"  INFO      LKH075
1 LAAIIOCN EQU   *-AAIIOCN               LENGTH OF "IOCNT"  INFO   LKH075
1          SPACE 2                                                  LKH075
1 *                                       END OF OUTPUT AREA        LKH075
1          MFPRE DNAME=AIAEND,ALIGN=F,PREFIX=A,MACID=AIA,MF=S,      :*R200C
1                DMACID=AIA                                         :*R200
2          CNOP  0,4
2 AAIAEND  DS    0F
```

```
1 LAAIAREA EQU   *-AAIAREA              LENGTH OF OUTPUT AREA     LKH075
1          SPACE 2                                                 LKH075
  MES2PAR  AINF  GLOBAL=D,IOCNT=D,P=B
1          #INTF INTNAME=AINF,REFTYPE=REQUEST,INTCOMP=001           GS 950
1 MES2PAR  MFPRE DNAME=AIAREA,MF=D,PREFIX=B,MACID=AIA,DMACID=AIA   :*R200
2 MES2PAR  DSECT ,
2                *,##### PREFIX=B, MACID=AIA #####
1 BAIAREA  DS    0F                      START OF OUTPUT AREA      LKH075
1          MFPRE DNAME=AIGLOB,ALIGN=F,PREFIX=B,MACID=AIG,MF=S,     :*R200C
1                DMACID=AIG                                        :*R200
2          CNOP  0,4
2 BAIGLOB  DS    0F
1 *                                      "GLOBAL" INFO PACKAGE     LKH075
1 BAIGTCPU DS    0FL8                    CPU TIME                  LKH075
1 BAIGCPUS DS    F                       CPU TIME SECONDS          LKH075
1 BAIGCPUN DS    F                       CPU TIME NANOSECONDS      LKH075
1 BAIG#IOS DS    F                       TOTAL # IO'S              LKH075
1 BAIG#BLK DS    F                       TOTAL # BLOCKS            GS 090
1 BAIGWSI  DS    FL8                     WORKING SET INTEGRAL      GS 090
1 BAIGLOBE EQU   *                       END OF "GLOBAL" INFO      LKH075
1 LBAIGLOB EQU   *-BAIGLOB              LENGTH OF "GLOBAL" INFO   LKH075
1          SPACE 2                                                 LKH075
1          MFPRE DNAME=AIIOCN,ALIGN=F,PREFIX=B,MACID=AII,MF=S :*R200DMACIC
1                D=AII                                        :*R200
2          CNOP  0,4
2 BAIIOCN  DS    0F
1 *                                      "IOCNT"  INFO PACKAGE     LKH075
1 BAII#IOS DS    F                       TOTAL # IO'S              LKH075
1 BAIIIOPD DS    F                # IO'S ON PUBLIC DEVICES         LKH075
1 BAIIIOSD DS    F                # IO'S ON SYSTEM PRIV. DISKS     LKH075

1 BAIIIOUD DS    F                # IO'S ON USER PRIVATE DISKS     LKH075
1 BAIIIOTP DS    F                # IO'S ON TAPE DEVICES           LKH075
1 BAIIIOUR DS    F                # IO'S ON UNIT RECORD DEVICES    LKH075
1 BAIIOCNE EQU   *                       END OF "IOCNT"  INFO      LKH075
1 LBAIIOCN EQU   *-BAIIOCN              LENGTH OF "IOCNT"  INFO   LKH075
1          SPACE 2                                                 LKH075
1 *                                      END OF OUTPUT AREA        LKH075
1          MFPRE DNAME=AIAEND,ALIGN=F,PREFIX=B,MACID=AIA,MF=S,     :*R200C
1                DMACID=AIA                                        :*R200
2          CNOP  0,4
2 BAIAEND  DS    0F
1 LBAIAREA EQU   *-BAIAREA              LENGTH OF OUTPUT AREA     LKH075
1          SPACE 2                                                 LKH075
           END
                 =X'F0'
```

Runtime log

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,ainf2), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,ainf2))
%  ASS6011 ASSEMBLY TIME: 407 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 106 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=ainf2
%  BLS0523 ELEMENT 'AINF2', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'AINF2', VERSION ' ' OF '<date> <time>' LOADED
MES1 successfully started ---------------------------------------------  (1)
MES2 successfully started ---------------------------------------------  (2)
%  IDA0199 PROGRAM BREAK AT ADDRESS X'0000B8', AMODE=24 -----------------  (3)
/copy-file from-file=oldexp,to-file=newexp ----------------------------  (4)
/resume-program
Elapsed Time used:  0000000000.247344000 ------------------------------  (5)
IOs before BKPT:              0000000000
IOs after BKPT:               0000000011
```

(1) MES1 was started; the macro executed without error.

(2) MES2 was started; the macro executed without error.

(3) Program execution was interrupted with **BKPT**.

Previously, MES2 was interrupted in order to save in AREA2A the measurements for CPU time, run time and number of inputs/outputs since starting. MES2 was then resumed.

(4) In the demonstration, a COPY-FILE command is issued (input/output). Program execution is then resumed. MES2 and MES1 are terminated. AREA2B contains the measurements for “global values” and the input/output count since starting MES2. The measurements for “time usage” and “input/output tally” cumulated since starting MES1 are stored in AREA1.

(5) For demonstration purposes, some of the values determined are displayed:

- runtime in seconds (calculated in MES1)
- input/output count at the point of interruption of MES2. The inputs/outputs generated by the COPY-FILE command are not taken into account.
- input/ouput count at the end of MES2, i.e including the inputs/outputs generated by the COPY-FILE command.

ALESRV – Connect task with/disconnect task from data space

General

Application area:	Extended addressing with data spaces; see page 61
Macro type:	Type S, MF format **3**: C/D/L/M/R/E form; see page 29

The **ALESRV** macro can can be used on all BS2000 servers
(see section “Extended addressing with data spaces” on page 61).

Macro description

The **ALESRV** connects a program running in AR mode with a data space that was set up using the **DSPSRV** macro. The system establishes a connection to the specified data space by searching for a free entry (ALE) in the task's access list, assigning it and returning the ALET for addressing purposes. When creating an ALE, the system checks whether the program is authorized to access this data space. Once the program has loaded a valid ALET into the access register, the system no longer performs a check before allowing access.

The **ALESRV** macro can also be used to release a data space. The system flags the entry (ALE) that connects the program to the data space as invalid in the task's access list. This entry is then available for a new connection (using FCT=CONNECT). If, during address conversion, an entry flagged as invalid is accessed, an interrupt occurs (see the **STXIT** macro).

If the user specifies an ALET, the **ALESRV** macro outputs the identification (SPID) associated with the data space which identifies the data space uniquely throughout the session.

The functions of the **ALESRV** macro make it possible to
- set up a connection between a program and a data space (FCT=CONNECT),
- clear this connection (FCT=DISCONN) and
- output the SPID of a data space (FCT=IDENTIFY).

Macro format and description of operands

ALESRV

FCT = { CONNECT, SPID=spid_addr
DISCONN, ALET=alet_addr
IDENTIFY, ALET=alet_addr }

,MF=C / D / L / M / R / E
[,PARAM=addr / (r)]
,PREFIX=N / p
,MACID=VDA / macid

FCT=
Specifies which of the functions of the **ALESRV** macro is to be executed.

CONNECT
Connects a program running in AR mode to an existing data space. A free entry (ALE) in the access list is assigned and the ALET is output.
The SPID operand must be specified.

DISCONN
Disconnects the program running in AR mode from the data space by releasing an assigned entry (ALE) in the access list, i.e. by flagging it as invalid.
The ALET operand must be specified.

IDENTIFY
Identifies a data space by means of its ALET, which points to an entry (ALE) in the access list. The SPID of the data space is output.
The ALET operand must be specified.

ALET=
Is the contents of an access register and points to an entry in the access list. This entry connects the program with the data space.
This operand may be an input or output operand.

alet_addr
Symbolic address (name) of a 4-byte field containing the ALET of a specific data space.

SPID=
Identifies a data space uniquely throughout the system. The system assigns the SPID on creation of a data space. This operand can be an input or output operand.

spid_addr
Symbolic address (name) of an 8-byte field containing the SPID of a data space.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29.
The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
It is possible to specify a PREFIX in the C form, D form, R form or M form of the macro, and additionally a MACID in the C form, R form or M form (see page 29).

Notes on the macro call

– The ALET operand can be an input or or output operand, i.e. FCT=CONNECT places it in the generated parameter list as an output operand; the next time this parameter list is used, the ALET operand is also valid as an input operand.

– Multiple connections to one and the same data space can be set up using different ALETs. Each of these ALETs must be deleted by a corresponding FCT=DISCONNECT call, since the system does not automatically delete all corresponding ALETs when releasing a data space.

– The ALET value assignment is deterministic, in other words for two programs in a BS2000 system run the sequence of the successful **ALESRV** calls with FCT=CONNECT and FCT=DISCONN is identical and parameterized in the same way (with CONNECT this means: SPID of the same data space specified; with DISCONN: the same ALET), thus both programs obtain the same ALET output value in the event of the nth ALESRV-CONNECT.
What is meant here is as follows:
If two TU programs behave in the same way with regard to connection to data spaces, they are also assigned the same ALETs.

– A data space is released using the **DSPSRV** FCT=DESTROY macro or on termination of the program that created the data space.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the ALESRV macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode)

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function executed successfully. MF=R can be used to read the following values from the parameter list: – if FCT=CONNECT: alet – if FCT=IDENTIFY: spid
X'02'	X'00'	X'0001'	Warning: the entry in the access list was deleted but the associated data space was already released (if FCT=DISCONN)
X'00'	X'01'	X'0003'	Error in data area
X'01'	X'01'	X'0004'	SPID operand missing (if FCT=CONNECT)
X'02'	X'01'	X'0004'	ALET operand missing (if FCT=DISCONN or FCT=IDENTIFY)
	X'20'	X'0005'	Internal error
X'00'	X'40'	X'0304'	Incorrect specification of SPID operand: the specified data space does not exist or the caller is not authorized to access this data space (if FCT=CONNECT)
X'01'	X'40'	X'0304'	Incorrect specification of SPID operand: the specified data space belongs to a different TU domain (if FCT=CONNECT)
X'00'	X'40'	X'0404'	Incorrect specification of ALET operand: the specified ALET is invalid or privileged (if FCT=DISCONN or FCT=IDENTIFY)
X'01'	X'40'	X'0404	Incorrect specification of ALET operand: the specified ALET references a data space in a different TU domain (if FCT=DISCONN or FCT=IDENTIFY)
X'00'	X'40'	X'0406'	Access list full, i.e. there are no free entries available for setting up a new connection between program and data space (if FCT=CONNECT)
X'00'	X'40'	X'0604'	Specified ALET references a data space that has already been deleted (if FCT=IDENTIFY)

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

For an **example** see section “Extended addressing with data spaces” on page 68.

ALINF – Request information on access lists

General

Application area:	Extended addressing with data spaces; see page 61
Macro type:	Type S, MF format **3**: C/D/L/M/E form; see page 29

The **ALINF** macro can be used on all BS2000 servers
(see section “Extended addressing with data spaces” on page 61).

Macro description

The **ALINF** macro tells the caller which ALETs (access list entry tokens) in the caller's task-specific access list correspond to a particular data space, which is identified uniquely by means of its SPID. If one or more ALETs are associated with a data space, the first of these is returned in the `<PREFIX><MACID>ALET` field (with RETURN=FIRST); on each consecutive call of the macro, the next ALET is returned (with RETURN=NEXT). The FROM operand can be used to specify where the search is to start, irrespective of any ALETS already found.

Macro format and description of operands

ALINF
SPID=spid_addr ,RETURN=FIRST / NEXT[,FROM=alet_addr / (r)] ,MF=C / D / L / M / E [,PARAM=addr / (r)] ,PREFIX=N / p ,MACID=VDI / macid

SPID=spid_addr
Identifies a data space uniquely throughout the system. The system assigns the SPID on creation of a data space.
spid_addr: Symbolic address (name) of an 8-byte field containing the SPID of the data space.

RETURN=
Specifies which of the ALETS associated with the data space is return in the `<PREFIX><MACID>ALET` field.

<u>FIRST</u>
The first ALET found that points to the data space identified by SPID is returned in the output field `<PREFIX><MACID>ALET`.

NEXT
The next ALET found is returned. The search starts at the ALET displayed in the output field `<PREFIX><MACID>ALET`. This ALET (≙ predecessor) is determined either by an **ALINF** call with RETURN=FIRST or by direct specification in the FROM operand. If the ALET output field does not already contain a predecessor as a result of a previous call and the FROM operand is not specified, the search starts at the next corresponding ALET in any location (any entry in the access list (ALE)).

FROM=
Specifies the ALET at which the search is to begin for the next corresponding ALET. The value is written to the output field `<PREFIX><MACID>ALET` and is read by the **ALINF** macro as an input operand (≙predecessor) if RETURN=NEXT. This operand may be specified only if MF=M.

alet_addr
Symbolic address of the ALET that is to be the predecessor.

(r)
Register containing the address value “addr”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see . The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

It is possible to specify a PREFIX in the C form, D form or M form of the macro, and additionally a MACID in the C form or M form (see page 29).
It is possible to specify a PREFIX in the C form, D form or M form of the macro, and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the ALINF macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode)

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function executed successfully. The ALET found can be read from the <pre> VDIALET field of the parameter list
X'00'	X'00'	X'0001'	No corresponding ALET found
X'00'	X'01'	X'0003'	Error in data area
X'07'	X'01'	X'0003'	Invalid RETURN operand
X'08'	X'01'	X'0003'	Invalid FROM operand: the operand value does not have ALET format
X'01'	X'01'	X'0004'	SPID operand missing
	X'20'	X'0009'	Internal error
X'01'	X'40'	X'0005'	Incorrect specification of SPID operand: – the specified SPID does not correspond to any data space or – the corresponding data space is privileged or – the data space belongs to a different domain
X'02'	X'40'	X'0005'	Incorrect specification of SPID operand: the corresponding data space was created by a different domain

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

For an **example** see section “Extended addressing with data spaces” on page 68.

AMODE31 – Query addressing mode

General

Application area:	XS programming; see page 164
Macro type:	Type O; see page 28

S line business server offer the user a choice between a 24-bit addressing mode and a 31-bit addressing mode. Any program being run above the 16-Mb boundary must execute in
31-bit addressing mode.

Macro description

The **AMODE31** macro informs the user of the address mode settings. The information is passed by setting the 2^{31} and 2^0 bits in the register specified. Bits 2^{30} through 2^1 are overwritten. The bit settings listed below mean:

Bit 2^{31}	Bit 2^0	Addressing mode
0	0	24-bit addressing mode (NXS)
1	0	31-bit addressing mode (XS)
1	1	32-bit addressing mode (servers with x86 architecture)

No return code indicating the macro execution is transferred.

Macro format and description of operands

AMODE31
reg

reg
Register where information is entered.

ARDS – Generate accounting records

General

Application area:	Accounting (system administration macro); see page 162 and 166
Macro type:	Definition macro; see page 28

Macro description

The macro **ARDS** generates a dummy section (DSECT) that serves to describe the structure of those accounting records which are created and written to the accounting file by the BS2000 accounting system. The accounting records are referred to by their record identifiers (e.g. UDAT, UACC, DSPC) and are described in the “Accounting records” manual [13].

A user wishing to obtain the structure of an accounting record that was created by a decoupled subsystem can request the structure of this record, depending on the current subsystem version.

The user can specify whether the macro describes the structure of all the accounting records, or a selection of them, or just a single record.

Macro format and description of operands

```
[name] ARDS

ALL / id / (id,id,...)
,DRV=OLD / NEW
,DSSM=OLD / NEW
,FT=OLD / NEW
,SPOOL=OLD / NEW
,UTM=OLD / NEW
,VM=OLD / NEW
,HSMS=OLD / NEW
```

name
Name of the generated DSECT. Default setting: name = ARDSECT.

ALL
The macro refers to all the accounting records.

id
Record identifier of the accounting record to be described.

(id,id,...)
List of record identifiers of the required accounting records.
The list must be specified in parentheses.

DRV=
Specifies the version-dependent structure of an accounting record that was created via the DRV subsystem (Dual Recording by Volume).

OLD
The structure is determined by the subsystem version that was known to the accounting system when the accounting record was transferred.
If no other version of the subsystem was transferred and loaded in the meantime, this structure is the same as the current one.

NEW
The structure is determined by the subsystem version that is currently loaded and is therefore always current.
If another subsystem version was transferred and loaded after transfer of the accounting records, the structure of the accounting records corresponds to the version that was unloaded (the “old” version) and not the “new” subsystem version that was loaded when **ARDS** was called. If NEW is specified, therefore, the caller must ensure that the current subsystem macro library (.GCLIB.UR) is known to the assembler (SET-TASKLIB command - see “Commands” manual [19], or COMPILE MACRO-LIBRARY=... statement in the compiler program - see “ASSEMBH” manual [2]).

DSSM=
Specifies the version-dependent structure of an accounting record that was created via the DSSM subsystem (Dynamic Subsystem Management).

OLD
Operand description as for DRV operand above.

NEW
Operand description as for DRV operand above.

FT=
Specifies the version-dependent structure of an accounting record that was created via the FT subsystem (File Transfer).

OLD
Operand description as for DRV operand above.

NEW
Operand description as for DRV operand above.

HSMS=
Specifies the version-dependent structure of an accounting record which was created via the HSMS subsystem (Hierarchical Storage Management System).

OLD
Operand description as for DRV operand above.

NEW
Operand description as for DRV operand above.

SPOOL=
Specifies the version-dependent structure of an accounting record that was created via the SPOOL subsystem (Simultaneous Peripheral Operation Online).

OLD
Operand description as for DRV operand above.

NEW
Operand description as for DRV operand above.

UTM=
Specifies the version-dependent structure of an accounting record that was created via the UTM subsystem (Universal Transaction Monitor).

OLD
Operand description as for DRV operand above.

NEW
Operand description as for DRV operand above.

VM=
Specifies the version-dependent structure of an accounting record that was created via the VM subsystem (Virtual Memory).

OLD
Operand description as for DRV operand above.

NEW
Operand description as for DRV operand above.

Notes on the macro call

- FT=NEW:may only be specified for an FT subsystem ≥ V5.0.
- UTM=NEW:may only be specified for a UTM subsystem ≥ V3.3.
- HSMS=NEW: The value NEW must be specified in order to obtain the DSECT for HSMS.
- The structure of an accounting record can be created only once for each assembler module, otherwise name conflicts occur.

AREC – Write user accounting record

General

Application area: Accounting; see page 162
Macro type: S-Typ, MF-Format **1**:
31-bit interface: standard/L/D/E form; see page 29

The accounting system compiles accounting data on the usage of data center resources and writes this data to the accounting file in the form of accounting records. The following are examples of accounting data:

– CPU time used, I/O data volume, working set integral, device and volume reservation data;
– system services used (user dumps etc.);
– memory reservation on pubsets.

The accounting file is evaluated using a special evaluation routine.
Basic structure of an accounting record:

`Record description`	Record ID, time stamp, ...
`User description`	User ID, account number, analyzed user task, ...
`Basic information`	Standard data
`Variable record portion`	Record extensions

A detailed description is provided in the “Accounting records” manual [13].

Macro description

The **AREC** macro initiates the writing of a user accounting record to the accounting file.

An accounting record can be one of the following:

– a UDAT accounting record with a record extension
– a UACC accounting record with a record ID
– an accounting record (freely) defined by the user

The DSECT operand enables the user to generate the basic structure for accounting records. For user-defined accounting records it is advisable to adhere to this structure.

Notes

- The user must use appropriate programs for evaluating accounting records (see the “Introduction to System Administration” manual [10]).
- System administration can limit the number of user accounting records per task for each user (MAX-ACCOUNT-RECORDS parameter in the user catalog). This limit applies to the whole command mode of a task (outside of program runs). Default value: MAX-ACCOUNT-RECORDS = 100; maximum of 100 accounting records in the command mode of a task.
- The writing of a freely defined accounting record to the accounting file requires the authorization MAX-ACCOUNT-RECORDS=NL (no limit).

Macro format and description of operands

```
AREC
-----------------------------------
 ⎧ DSECT=RECORD          ⎫
 ⎨ ,DATA=adr / (r)       ⎬
 ⎪ ,ID=adr / (r)         ⎪
 ⎩ ,RECORD=adr / (r)     ⎭

,MF=S / L / (E,...) / D

,P=I / p
```

DSECT=RECORD
A dummy section (DSECT) that reproduces the basic structure of an accounting record is generated.
A prefix P (P = 1 letter) may be specified.
The MF operand may not be specified.

DATA=
This describes the address of the record extension (data string) that is entered in the UDAT accounting record. The length of the following text should be specified in bytes 0-1 of the data string.
Length of text ≤ 255 bytes.

addr
Symbolic address (name) of the field containing the data string.

(r)
Register containing the address value “addr”.

ID=
This describes the address of a character string (data string) that is entered as a record ID in the UACC accounting record.
Length of the character string ≤ 8 bytes.

addr
Symbolic address (name) of the field containing the data string.

(r)
Register containing the address value “addr”.

RECORD=
This describes the address of the accounting record (freely) defined by the user.
The record ID and the user description must also be supplied with values by the user in the accounting record.
Record length ≤ 496 bytes.

The operand is only permissible for users with MAX-ACCOUNT-RECORDS=NL (in the user catalog).

addr
Symbolic address (name) of the field containing the accounting record.

(r)
Register containing the address value “addr”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

If MF=D, a prefix P (P = 1 letter) can be specified, as shown in the macro format.

Return information and error flags

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the AREC macro is transferred in register R15.

X'bb'	**X'aa'**	**Meaning**
X'00'	X'00'	Function executed normally
X'04'	X'00'	The record was not written because accounting was deactivated
X'08'	X'00'	The record was not written because the relevant record type was deactivated
X'00'	X'04'	Address error (parameter list)
X'00'	X'08'	Invalid function code (for SVC 99 or for AREC)
X'00'	X'0C'	The function is not allowed for this user
X'00'	X'10'	Address error (DATA, ID or RECORD operand)
X'00'	X'14'	Invalid record ID
X'00'	X'18'	Max. record length exceeded
X'00'	X'1C'	Number of permissible records exceeded
X'00'	X'20'	Invalid register
X'00'	X'24'	System or resource error; more information in the secondary return code
X'04'	X'24'	No memory available
X'08'	X'24'	No job information
X'0C'	X'24'	CLTF error: error on writing a record

Basic structure of an accounting record

```
           AREC  DSECT=RECORD
1          MFCHK DNAME=ARLDS,MF=D,PREFIX=I,MACID=ARL,DMACID=ARL
2 IARLDS   DSECT ,
2                *,##### PREFIX=I, MACID=ARL #####
1          DS    OF                      AREC PARAMETER LIST
1 ****************************************************************
1 *             GENERAL LAYOUT OF ACCOUNT    RECORDS             *
1 ****************************************************************
1 IARLRHDR DS    OF                      RECORD HEADER
1 IARLRL   DS    Y                       RECORD LENGTH
1          DS    XL2                     RESERVED
1 IARLID   DS    CL4                     RECORD ID
1          DS    FL8                     RESERVED (TIME STAMP)
1 IARLLUS  DS    Y                       LENGTH OF USER HEADER
1 IARLLBI  DS    Y                       LENGTH OF BASIC INFORMATION
1          DS    XL4                     RESERVED
1 *
1 *
1 IARLUSER DS    OF                      USER HEADER
1 IARLUSID DS    CL8                     USER IDENTIFICATION
1 IARLACNT DS    CL8                     ACCOUNT NUMBER
1 IARLTSN  DS    CL4                     TSN
1 *
1 *
1          ORG   IARLID+X'1000'
1 IARLBI   DS    OF                      BASIC INFORMATION
1 *
1 *
1          ORG   IARLID+X'2000'
1 IARLEXT  DS    OH                      VARIABLE EXTENSION PART
1 IARLEXTH DS    OH                      EXTENSION HEADER
1 IARL#EXT DS    H                       NUMBER EXTENSIONS
1 IARLDEXT DS    OH                      START LIST OF DISTANCES
1 *
1          ORG   IARLID+X'3000'
1 IARLSEXT DS    OH                      STRING EXTENSION
1 IARLSEXI DS    CL2                     EXTENSION ID
1 IARLLSEX DS    Y                       LENGTH OF STRING
1 IARLSEXB DS    OC                      BEGIN OF STRING
1 *
1          ORG   IARLID+X'3000'
1 IARLAEXT DS    OH                      ARRAY EXTENSION
1 IARLAEXI DS    CL2                     EXTENSION ID
1 IARLAEX# DS    X                       NUMBER OF ELEMENTS
1 IARLAEXL DS    X                       LENGTH OF ONE ELEMENT
1 IARLAEXB DS    OH                      BEGIN OF FIRST ELEMENT
```

ASHARE – Load user's shared code into common memory pools

General

Application area:	Linking and loading; see page 47
Macro type:	S-Typ, MF-Format **2**: Type S, MF format **2**: standard/C/D/L/E/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **ASHARE** macro links and loads the user's shared code into a common memory pool. This shared code may consist of a set of modules (see section “Common memory areas shared by several users (Memory pools)” on page 55). Any user connected to the common memory pool can access this type of shared program via its program name or via all other nonmasked CSECTS or ENTRYs, using the LOAD-EXECUTABLE-PROGRAM and START-EXECUTABLE-PROGRAM commands (or LOAD-PROGRAM and START-PROGRAM) or the **BIND** and **VSVI1** macros.

In indirect linking, **ASHARE** can be used for loading server modules into a common memory pool.

Macro format and description of operands

```
ASHARE

[,{ CONTEXT=name        }]
  { CONTXT@=addr / (r)  }

  { LIBNAM=file / *     }
[,{ LIBNAM@=addr / (r)  }]
  { LIBLINK=name        }
  { LIBLNK@=addr / (r)  }

,ALTLIB=*DBLOPT / NO / YES

,INTVERS=BLSP2 / SRV001

,MAP=*DBLOPT / NO / BOTH / (BOTH,nn) / nn / SYSOUT

[,MPID=addr / (r)]

[,MSGCTRL=*DBLOPT / INFORMATION / WARNING / ERROR / NONE]

,{ PGMVERS=*STD / version }
 { PGMVER@=addr / (r)     }

[,{ PROGRAM=name      }]
  { PROG@=addr / (r)  }

[,{ REPFILE=file        }]
  { REPFIL@=addr / (r)  }

,RESMP=NO / YES

,RESSYS=NO / YES

[,START@=addr / (r)]

,{ SYMBOL=name        }
 { SYMBOL@=addr / (r) }

,SYMTYP=ANY / CSECT / CSEN / MODULE / ENTRY

[,{ VERSION=version  }]
  { VERS@=addr / (r) }

,MF=S / C / D / E / L / M

[,PARAM=addr / (r)]

,PREFIX=P / p

,MACID=BAS / macid
```

The operands are described in alphabetical order below.

ALTLIB=
Specifies whether alternate libraries are to be searched for the object defined by SYMBOL or SYMBOL@. Alternate libraries are assigned using the file link name BLSLIBnn (00≤nn≤99) or $BLSLBnn (for alternate system libraries) and are also used for the autolink function of DBL.

<u>*DBLOPT</u>
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, ALTLIB=NO applies.

NO
Alternate libraries will not be searched.

YES
Alternate libraries will be searched.

i The operands ALTLIB=YES and LIBNAM/LIBNAM@/LIBLINK/LIBLNK@ may be specified together in the same macro. If, however, the main library specified by LIBNAM/LIBNAM@/LIBLINK/LIBLNK@ is not present, DBL will abort the processing. ALTLIB=YES cannot serve as a substitute for a missing or invalid main library.

CONTEXT=
Specifies the name of the context in which the program is to be loaded. If this context already exists in a memory pool that is accessible by the user task, this name must refer to the memory pool specified in the MPID operand. Up to 15 contexts may be defined in one memory pool.

name
Context name. The first character of the context name must be “#”. The name may be up to 32 characters long. The default name is “#” followed by the first 31 characters of the memory pool specified in the MPID operand.

CONTXT@=
Specifies the address of a field containing the context name. Can be specified only if MF=M.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

INTVERS=
This operand defines the version of the macro interface ASHARE.

BLSP2
Default. Corresponds to macro version 2.

SRV001
Corresponds to macro version 3. This version is supported as of BLSSERV V2.4A.

LIBLINK=name
Explicit specification of the file link name of the main library. This name may be up to 8 characters long.

LIBLNK@=
Specifies the address of a field containing the file link name of the main library. May be specified only if MF=M.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

LIBNAM=
Specifies the main library in which the search for the object defined with SYMBOL or SYMBOL@ is to take place. The main library can be defined by specifying its file name explicitly or by means of a file link name. The EAM object module file is defined using the file name “*”. It is not possible to specify a file link name for the EAM object module file. If the LIBNAM, LIBNAM@, LIBLINK and LIBLNK@ operands are omitted, the library with the file link name BLSLIB is searched.

The main library is searched *before* the alternate libraries (see page 264). It is also used for the autolink function of DBL. If the LIBLINK or LIBLNK@ operand is specified, any LIBNAM or LIBNAM@ entry is ignored.

file
Explicit specification of the file name of the main library. The file name may be up to 54 characters long.

*****
Specifies the EAM object module file as the main library.

LIBNAM@=
Specifies the address of a field containing the name of the main library. May be specified only if MF=M.

addr Address of an auxiliary field which contains the field address searched for.

(r) r = register containing the field address searched for.

MAP=
Specified only with INTVERS=SRVxxx and xxx ≥ 001
Defines whether or not a DBL map is output and specifies the output destination for the DBL map.

*DBLOPT
This operand value is taken over from the last call of the MODIFY-DBL-DEFAULTS command. If no value has yet been defined with MODIFY-DBL-DEFAULTS for the operand involved, MAP=NO applies.

NO
No DBL map is output.

BOTH
The output destination is the SYSOUT system file and the SYSLST00 system file.

(BOTH,nn)
The output destination is the SYSOUT system file and a SYSLSTnn system file (00≤nn≤99).

nn
The output destination is a system file from the range SYSLST00 through SYSLST99 whose number must be specified here.
The number must be specified as a 2-digit number (00 for 0 etc.).

SYSOUT
The output destination is the SYSOUT system file.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM, PREFIX and MACID, see section "S-type macros" on page 29. The valid MF values are given at the start of the macro description under "Macro type" and are included in the macro format.

It is possible to specify a PREFIX (consisting of one alphabetic character) in the C form, D form or M form of the macro, and additionally a MACID (three alphabetic characters) in the C form or M form (see section "S-type macros" on page 29).

MPID=addr
Symbolic address of a 4-byte field containing the identifier of a memory pool into which the load unit will be loaded. This identifier is made available to the user by means of the **ENAMP** macro, which must be executed before the **ASHARE** macro.

(r)
r = register containing the address value "addr". May be specified only if MF=M.

MSGCTRL=
Specifies the lowest message class; DBL messages at and above this level will be output. The value set in the load call with LOAD-EXECUTABLE-PROGRAM or START-EXECUTABLE-PROGRAM (or LOAD-PROGRAM or START-PROGRAM) will be used as the default value.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, MSGCTRL=INFORMATION applies.

INFORMATION
All classes of message will be output.

WARNING
Only messages of the WARNING and ERROR classes will be output.

ERROR
Only messages of the ERROR class will be output.

NONE
No DBL messages will be output.

PGMVERS=
Specifies the program version.

<u>*STD</u>
The load unit resulting from the load call receives the version of the loaded library element as the program version. If the symbol specified in the load call is already loaded, a search is carried out for the program version that was set using the SELECT-PROGRAM-VERSION command. If no program version has been set, DBL uses the first symbol found.

version
The version specification may be up to 24 characters long. If this program version already exists in the common memory pool, the load procedure is aborted and the corresponding return code is output.

PGMVER@=
Specifies the address of a field containing the program version. May be specified only if MF=M.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

PROGRAM=
Identifies the program.

name
The specified name must be unique and must not be more than 32 characters long. The default value is the name specified for SYMBOL or SYMBOL@. If this program already exists in a common memory pool that can be accessed by the user, the load procedure is aborted.

PROG@=
Specifies the address of a field containing the program name. May be specified only if MF=M.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

REPFILE=
Specifies the name of the REP file which contains REP records in standard BS2000 format (see the “Introduction to System Administration” manual [10]). If there is an error during the processing of REP records, DBL outputs an error message on SYSOUT and the errored REP record is skipped. REP processing then resumes.

file
The file name may be up to 54 characters long.

REPFIL@=
Specifies the address of a field which contains the name of a REP file. May be specified only if MF=M.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

RESMP=
Specifies the scope for resolving external references in the shared code in the memory pool.

NO
Only the context specified in the CONTEXT operand is used to resolve external references.

YES
All contexts that refer to the memory pool are used to resolve external references.

RESSYS=
Specifies whether the shared code in the system address space (class 3/4/5 memory) is also to be searched in order to resolve external references. This shared code is loaded with DSSM in the form of nonprivileged subsystems.

NO
Shared code in the system address space is not used to resolve external references.

YES
Shared code in the system address space is used to resolve any external references that were not resolved as a result of searching the shared code in the memory pool.

START@=addr
Symbolic address of a 4 byte field to which DBL transfers the start address of the load unit in the memory pool. The field must be aligned on a word boundary. The user must request the start address explicitly with START@; it is *not* returned by default.

(r)
r = register containing containing the address value “addr”. May be specified only if MF=M.

SYMBOL=name
Explicit specification of an object name. DBL uses this name to determine which module of the load unit is to be loaded into the memory pool first. The name can refer to the following objects:

- control section (CSECT),
- entry point (ENTRY),
- object module (OM) (element name),
- linking loader module (LLM) (element name).

The name may be up to 32 characters long. The type of the object is defined using the SYMTYP operand.

SYMBOL@=
Specifies the address of a field containing the name of the object. May be specified only if MF=M.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

SYMTYP=
Specifies the type of object defined with the name SYMBOL or SYMBOL@ and defines the search sequence for the object. A symbol (CSECT or ENTRY) or module (OM or LLM) can be defined as the type of object.
If the object is a symbol, the name SYMBOL or SYMBOL@ designates a symbol name and can be

- the name of a nonmasked CSECT or ENTRY entry in a program library (type R or type L),
- the name of a nonmasked CSECT or ENTRY entry in an object module library (OML) or in the EAM object module file.

If the object is a module, the name SYMBOL or SYMBOL@ designates a module name and can be

- the name of a library element (type R or type L) in a program library or
- the name of a library element in an object module library (OML).

ANY
The search is performed in the following sequence:

1. LLMs with the module name SYMBOL or SYMBOL@
2. OMs with the module name SYMBOL or SYMBOL@
3. Symbols with the symbol name SYMBOL or SYMBOL@ in an LLM. DBL searches for CSECTS first. If no CSECT is found, it searches for ENTRYs.
4. Symbols with the symbol name SYMBOL or SYMBOL@ in an OM. DBL searches for CSECTS first. If no CSECT is found, it searches for ENTRYs.

CSECT
Only control sections (CSECTs) with the symbol name SYMBOL@ or SYMBOL are searched for.

ENTRY
Only entry points (ENTRYs) with the symbol name SYMBOL or SYMBOL@ are searched for.

CSEN
CSECTs *and* ENTRYs with the symbol name SYMBOL or SYMBOL@ are searched for. DBL searches for CSECTs first. If no CSECT is found, it searches for ENTRYs.

MODULE
Only modules with the module name SYMBOL or SYMBOL@ are searched for.

VERSION=
Specifies the element version of the library element defined by SYMBOL or SYMBOL@. If SYMTYP=ANY, the VERSION operand is taken into account only if the object name (SYMBOL or SYMBOL@) refers to a module in a program library (type R or type L). If a CSECT or ENTRY name is specified, the VERSION operand is ignored. If the operand is not specified, the default value for the highest element version in program libraries is used (see the “LMS” manual [29]).

version
Explicit specification of the element version. The version specification may be up to 24 characters long.

VERS@=
Specifies the address of a field containing the element version. May be specified only if MF=M.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

Notes on the macro call

- Before calling the **ASHARE** macro, the user must have been connected to the common memory pool by means of the **ENAMP** macro. The memory pool must have been set up with the **ENAMP** operand FIXED=YES and the SCOPE may not be LOCAL.
- If the memory pool was created in address space higher than 16 Mbytes, all CSECTS of the loaded modules must have the attribute RMODE=ANY.
- External references that cannot be resolved and name conflicts are not permitted.
- LLMs with user-defined slices cannot be loaded into common memory pools using the **ASHARE** macro. In the case of LLMs whose slices have been defined with the PUBLIC attribute, only the PUBLIC part is loaded using **ASHARE**. This also applies to the autolink function of DBL.
- Test and diagnostic information (LSD) is not taken into account.
- The number of memory pools in which the user can store shared code is limited to 16 per scope (ENAMP operand SCOPE) for each user ID:

  1. up to 16 memory pools for SCOPE=GROUP,
  2. up to 16 memory pools assigned to a specific user group number for SCOPE=USER-GROUP and
  3. up to 16 memory pools for SCOPE=GLOBAL.

  A task can therefore access a total of 48 memory pools for shared code in the user address space. Shared code that is to be accessible for all users as in SCOPE=GLOBAL, can also be loaded into the system address space as a nonprivileged subsystem with DSSM.

- If the specified program version has not yet been loaded but a program with this name already exists in the link context (see CONTEXT parameter), loading is rejected because of the name conflict.
  To avoid such name conflicts, different versions of a program must be loaded into different contexts.
- Only a limited number of memory pages in the system address space are available for storing linking and loading information concerning the modules loaded with **ASHARE**. The number of pages is defined in the startup parameter service via the class 2 system parameter BLSUSLIM (see the “Introduction to System Administration” manual [10]).
- A resident memory pool can be used for **ASHARE** only if it is created in a load module that was loaded with the corresponding value for #RESIDENT-PAGES (see the START-EXECUTABLE-PROGRAM command [19]).
- When searching for the primary input, symbols of the FILE type are ignored for **ASHARE** since an FILE symbol may never be used as the entry point of a server module for indirect linking.

Return information and error flags

The start address of the load unit is transferred to the field specified with the START@ operand.

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the ASHARE macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function executed normally
X'00'	X'01'	X'0001'	Symbol name omitted
X'00'	X'01'	X'0002'	Memory pool identifier (MPID operand) omitted
X'00'	X'01'	X'0003'	Invalid identifier specified in MPID
X'00'	X'01'	X'0004'	Invalid context name
X'00'	X'01'	X'0005'	Invalid symbol type
X'00'	X'01'	X'0006'	Invalid field address specified for START@
X'00'	X'01'	X'0007'	Invalid memory pool name.
X'00'	X'01'	X'0008'	Invalid MAP operand.
X'00'	X'01'	X'0009'	The SYSLST number in the MAP operand is invalid.
X'00'	X'01'	X'0011'	Task not connected to memory pool
X'00'	X'01'	X'0012'	Memory pool created with wrong attributes. FIXED=YES missing from ENAMP macro or CLASS=5 or SCOPE=LOCAL specified in ENAMP

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'01'	X'0013'	Program name specified for PROGRAM not unique; already exists in a memory pool
X' 00'	X'01'	X'0014'	Context specified for CONTEXT already exists in another memory pool
X'00'	X'01'	X'0015'	Context cannot be generated since the maximum number of 15 contexts has already been reached in this memory pool
X'00'	X'01'	X'0016'	Maximum number of 16 memory pools with this scope already in use for this user ID. No more memory pools can be used with this scope
X'00'	X'01'	X'0017'	System storage area for linking and loading information is full (number of memory pages defined in system parameter BLSUSLIM already reached)
X'00'	X'01'	X'0018'	Specified symbol already loaded in this context
X'00'	X'01'	X'0019'	Memory pool already specified in a BIND macro and can no longer be used for shared code
X'00'	X'01'	X'0020'	Error during linking/loading. The return code relating to this errored load procedure is transferred in the parameter list. Possible values are described with the BIND macro
X'00'	X'20'	X'0100'	System error
X'00'	X'20'	X'0101'	Internal DBL error
X'00'	X'20'	X'0103'	DBL Lock Manager error during processing of ASHARE macro
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported
X'00'	X'03'	X'FFFF'	The interface version is not supported

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

ASPC – Enter memory allocation

General

Application area:	Accounting/system administration macro; see page 162
Macro type:	Type S, MF format **1**: 31-bit interface: standard/L/D/E form; see page 29

The **ASPC** macro can only be called under the TSOS (system administration) ID. An unauthorized call will be rejected with return code X'0C'.

Macro description

The **ASPC** macro is used to record the current memory allocation on public and private volumes. The PAM page allocation is recorded in the form of accounting records.

The **ASPC** macro writes one or more accounting records (space stocktaking records: record ID = DSPC or DSPP) to the accounting file for every locally available public volume set (PVS) or private disk (PD). Each record contains the following:

– the catalog ID of the PVS
– time of stocktaking (date, time of day);
– an indicator relating to the completeness of the record:
  `'C'`: continuation character
  `'L'`: recording complete ('L' = last)
  `'I'`: recording incomplete (the PVS was exported during recording)
– user IDs for the PVS (up to 26 user IDs per record);
– number of PAM blocks allocated per user ID;

For a detailed description of DSPC and DSPP records, see the “Accounting records” manual [13]. Several records are written for one PVS if the PVS has more than 26 user IDs. In this case, the records contain the continuation character.
ASPC has no operands that control functions.

Macro format and description of operands

ASPC
,MF=S / L / (E,..) / D ,P=I / p

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
If MF=D, a prefix P (p = 1 letter) can be specified, as shown in the macro format.

Return information and error flags

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the ASPC macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'00'	X'00'	Normal execution
X'04'	X'00'	The record was not written because accounting or the DSPC record was deactivated
X'00'	X'04'	Address error (operand list)
X'00'	X'08'	Invalid function code (for SVC 99 in the operand list)
X'00'	X'0C'	User is not authorized to call the macro
X'00'	X'10'	System or resource error
X'04'	X'10'	No job information
X'08'	X'10'	Error on accessing MRSCAT
X'0C'	X'10'	Error on accessing the internal USERTABLE
X'10'	X'10'	Error on writing a record

In all cases where aa = X'10' there is also an entry in the SERSLOG file containing the operand list and the return code of the called system function.

AUDIT – Control audit mode

General

Application area:	Debugging aids; see page 162
Macro type:	Type S, MF format **1**: standard/L/D/E form; see page 29

No symbolic names are generated in the data area in the case of MF=L.

> The function harware AUDIT (see below) is only available on servers with /390 architecture.
> On other servers the function does return RC=0, but it is not executed.

Macro description

The **AUDIT** macro offers the user functions for program auditing.

The **linkage AUDIT** is available in addition to the **hardware AUDIT**. Both these functions permit tracing of the program run by recording either the addresses of branches performed (hardware AUDIT) or the destination addresses of subprogram branches (linkage AUDIT). The two functions are mutually independent.

The hardware AUDIT enters the source addresses for each branch performed in an AUDIT table, whereas the linkage AUDIT enters the branch destination addresses in the AUDIT table on execution of the BASR, BALR, BASSM and BAKR commands. The linkage AUDIT thus makes it possible to record all the subprograms called within the program run, provided that they were called using one of the above commands.

Separate AUDIT tables are created for the hardware and linkage AUDIT and identified accordingly in the header line on output. An AUDIT table consists of 64 word-length entries for the hardware AUDIT and 1024 entries for the linkage AUDIT and is cyclically overwritten unless specified otherwise. The macro may refer to the entire run of a task or all tasks or be restricted to one process within its own task (e.g. contingency process). The linkage AUDIT may also be processor-local.

It is possible to permit or prohibit the hardware AUDIT and the linkage AUDIT

– For an entire session
  Control is achieved via the system parameter AUDALLOW=YES/NO.
  If any local linkage AUDIT has been activated, it is deactivated if the system parameter AUDALLOW=NO has been set.
  The hardware AUDIT and the linkage AUDIT can only be controlled in combination.

– For a user ID

Control is performed using the operand HARDWARE-AUDIT or LINKAGE-AUDIT= *UNCHANGED / *NOT-ALLOWED / *ALLOWED in the MODIFY-USER-ATTRIBUTES or ADD-USER command.

- For a task
  Control is performed using the operand HARDWARE-AUDIT or LINKAGE-AUDIT= *UNCHANGED / *NOT-ALLOWED / *ALLOWED in the MODIFY-TEST-OPTIONS command.

The **AUDIT** macro is rejected with return code X'81003C' if the hardware and linkage AUDITs are not permitted.

Comments on the local linkage AUDIT

The linkage AUDIT is used for the CPU-specific activation of the linkage AUDIT for all the active CPUs or all the logical machines of a server configuration. For each CPU, a trace table is created in privileged class 3 memory and this is maintained throughout the entire session. The linkage AUDIT can be activated or deactivated via the parameter service during the startup phase. System administration can use the /START-LINKAGE-AUDIT/STOP-LINKAGE-AUDIT commands and the **AUDIT** macro to activate/deactivate the local linkage AUDIT during the session.

Macro format and description of operands

```
AUDIT

FCT=HWA / LNKA
,SCOPE=TASK / FUNCT / ALLTASK / SIHGLOB / SYSGLOB
,STATE=USER / SYS / PROC

          ⎧ ON [,SAVE=n]         ⎫
          ⎪ OFF                  ⎪
,ACTION=  ⎨ CONT                 ⎬
          ⎪ DISC                 ⎪
          ⎩ GET [,TABLE=addr]    ⎭

[,⎧ TID=tid ⎫]
  ⎩ TSN=tsn ⎭

[,PARMOD=31]
,MF=S / L / (E,..) / D
,ID=AUD / pre
```

The operands are described in alphabetical order below.

ACTION=

Specifies the desired AUDIT operation: activate/deactivate, interrupt, continue or output.

ON

The 64-word (256-byte) or 1024-word (4096-byte) AUDIT table is created (if not already present), and initialized by overwriting any existing entries with binary zero. The current pointer is set to the beginning of the table and AUDIT mode is activated.
If the AUDIT table already exists, the current pointer is set after the last entry or, if the table is already full, to the beginning of the table (cyclic overwriting), and AUDIT mode is activated.

OFF

Deactivates AUDIT mode and releases the AUDIT table and the save table, if present. If the AUDIT table was copied via the GET operand to a memory area designated by the user, this area is retained until the program is terminated. The SCOPE=FUNCT operand may not be specified; the SAVE operand is ignored.

CONT

Reactivates the AUDIT following a preceding ACTION=DISC call. Auditing to the AUDIT table is resumed from the point at which it was interrupted by the ACTION=DISC call. If a save table is present it continues to be used for saving the AUDIT tables.

This operand can only be specified in conjunction with SCOPE=TASK. If no AUDIT is active at the time of the CONTINUE call, it is activated for the specified processor state (USER or SYS) for the entire task. The operands TID and TSN may not be specified. The SAVE operand is ignored.

DISC

Deactivates AUDIT mode while retaining the AUDIT table and the save table, if present. The DISCONTINUE function can be used only for the entire run of the user's own task (SCOPE=TASK); the operands TID and TSN may not be specified; the SAVE operand is ignored.

GET

Only permitted if SCOPE=TASK; the SAVE operand is ignored

For FCT=HWA:

The 64-word hardware AUDIT table (excluding the save table) is copied to the memory area designated by the user by means of the virtual address specified in the TABLE operand.
Unlike the SHOW command, the contents of the AUDIT table are read in in unchanged chronological order.

For FCT=LNKA:

All the linkage AUDIT trace information (AUDIT and save table) is copied to the memory area designated by the user in the linkage AUDIT data area by means of the virtual address `audTAB` and the length `audLBUF` The last AUDIT table entry appears at the beginning of the output.

In the linkage AUDIT, the address and length of the transfer buffer must be stored in the `audTAB` or `audLBUF` fields. Note that the address and length of the transfer buffer cannot be specified directly in the macro; instead they must be entered in the parameter list using the L form. In addition to the AUDIT table, the address of the next free entry in the transfer buffer is returned in the `audTABE` field. If the buffer is full, the end address of the buffer is returned. If the interface is called with ACTION=GET and the value X'00000000' in the `audTAB` field, the buffer size for a subsequent GET call is returned in the `audLBUF` field. If the transfer buffer specified in the `audTAB` field is too small, the request will only be partially carried out. The linkage AUDIT table is transferred only with the length specified in the `audLBUF` field. The return code X'400020' is returned in addition.

FCT=
This operand specifies the AUDIT function to which the following parameters refer.

HWA
The AUDIT macro refers to the **hardware AUDIT**.

LNKA
The AUDIT macro refers to the **linkage AUDIT**.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
If MF=D or MF=L, a prefix ID (pre = 1..3 letters) can be specified, as shown in the macro format.

PARMOD=
This parameter is unnecessary and is still supported for compatibility reasons only.
In general only the 31-bit interface is generated regardless of the operand specified.

Note: 24-bit coding that has already been generated will, of course, remain executable.

SAVE=
Creates a save area (save table) for the AUDIT table in the privileged class 5 memory of that task which is to be monitored with **AUDIT** (with TU), or in the privileged class 3 memory (with TPR). This save area accepts the contents of the AUDIT table before it is cyclically overwritten. This increases the requestable amount of AUDIT information.
This operand is effective only if specified in conjunction with ACTION=ON and only if no AUDIT table is yet available (e.g. after ACTION=OFF).

n
Specifies the number of 4K pages for the save area.
With nonprivileged logging of the processor state TU, integer values of 0 ≤ n ≤ 16 may be specified for n. A maximum of 64K can be requested for the save area, corresponding to the size of 256 hardware AUDIT trace tables or 16 linkage AUDIT trace tables.

With privileged logging of the processor state TPR using the hardware AUDIT, only the values 0 and 1 are meaningful for n; greater values (n > 1) are replaced by the value 1. A maximum of 4KB can therefore be requested for the save area for TPR in the hardware AUDIT, corresponding to the size of 16 AUDIT trace tables.
This operand may not be specified for privileged logging of the processor state TPR using the linkage AUDIT or if SCOPE=SIHGLOB/SYSGLOB is specified.

SCOPE=
Specifies the program area to be logged.

Note
An AUDIT job of a very large program area or a higher PCB replaces that of a smaller program area or a lower PCB (e.g. TASK replaces FUNCT), but not vice versa.

TASK
The requesting task or the task specified in the TID or TSN operand is to be logged.

FUNCT
The PCB-specific function to be logged is that function which refers to the highest interrupted PCB of the processor state specified in the STATE operand. FUNC is permitted for the user's own task only and if ACTION=ON is specified. The operands TID and TSN may not be specified.

ALLTASK
All branch addresses belonging to the processor state TPR are to be logged with all tasks. If the TID or TSN operands are listed at the same time this leads to an error message. The SAVE=n entry is ignored if FCT=HWA is specified. SAVE=n may not be specified if FCT=LNKA (see SAVE operand). If necessary, ALLTASK first switches off an active task-wide hardware audit in every task and then switches it back on without the save table.
If the task-wide TPR linkage AUDIT is already switched on for a task, it is not changed by the ALLTASK function.
If a task-wide hardware audit is inactive, it is not switched off and the save table is retained. If new tasks are added, the AUDIT is generally activated for these without the save table.
This value can only be entered under the system administration ID (TSOS).

SIHGLOB
All branch addresses of the processor state SIH are logged in a processor-local AUDIT table. This specification is permitted only in conjunction with FCT=LNKA, STATE=PROC and ACTION=ON/OFF. The operands TID, TSN and SAVE may not be specified.
This value can only be entered under the system administration ID (TSOS).

SYSGLOB
All branch addresses of the processor states SIH and TPR are logged in a processor-local AUDIT table. SCOPE=SYSGLOB may only be used if no privileged linkage AUDIT is activated in any task throughout the system or no TPR linkage AUDIT is in the 'DISCONTINUE' state. This specification is permitted only in conjunction with FCT=LNKA, STATE=PROC and ACTION=ON/OFF. The operands TID, TSN and SAVE may not be specified.
This value can only be entered under the system administration ID (TSOS).

STATE=
Processor state (TU, TPR or SIH) to which the scope specified in the SCOPE operand refers. If more than one processor state is to be logged simultaneously, the macro must be called a number of times.

USER
The scope specified in the SCOPE operand refers to the processor state TU (USER). The operand value P1 is still supported for reasons of compatibility and for the hardware AUDIT only. Default for SCOPE=FUNCT and SCOPE=TASK.

SYS
The scope specified in the SCOPE operand refers to the processor state TPR (SYS). The operand value P2 is still supported for reasons of compatibility and for the hardware AUDIT only. Default and mandatory for SCOPE=ALLTASK.
This value can only be specified under the system administration ID (TSOS).

PROC
The scope specified in the SCOPE operand is processor-local and refers to the SIH processor state or SIH and TPR. This specification is permitted only in conjunction with FCT=LNKA. Default and mandatory for SCOPE=SIHGLOB/SYSGLOB.
This value can only be specified under the system administration ID (TSOS).

TABLE=
Specifies the address to which the contents of the hardware AUDIT table are to be written. This may only be specified for hardware AUDIT and in conjunction with ACTION=GET.

addr
Virtual memory address which must point to a previously assigned memory area with write access authorization. Address 0 must not be specified.
(addr can be specified by 1 to 8 hexadecimal digits or by the appropriate number of decimal digits).

It is helpful to enter only a dummy value for the TABLE operand and then to overwrite the `audTAB` with the appropriate address dynamically in the program.

In the linkage AUDIT the transfer of AUDIT trace information to a user area is controlled exclusively by the GET operand.

TID=
Specifies the task which is to be monitored with AUDIT by its internal task number. The operand may only be specified together with SCOPE=TASK (see also the note for the TSN operand).

tid
Internal task number; can be specified in the followinf form:

h[hhhhhhh]:	1-8 hexadecimal digits, which the system pads out to 8 places with leading zeros as necessary.

TSN=
Specifies the task which is to be monitored with **AUDIT** by its job number.

tsn
Job number (TSN); can be specified as follows:

n[nnn]:	1-4 digits, which the system pads out with leading zeros to 4 places as necessary
a[aaa]:	1-4 alphanumeric characters, which the system pads out with leading zeros to 4 places as necessary
c'a[aaa]':	1-4 characters, which the system pads out with leading zeros to 4 places as necessary (user-specified blanks are retained)

Notes

– If neither of the two operands TID or TSN is specified, the AUDIT request applies to the requesting task itself.
The TID or TSN operand may be specified only in conjunction with SCOPE=TASK.
The ACTION=DISC and ACTION=CONT operands are not permitted in conjunction with the TID or TSN operand.

Return information and error flags

R15:

0	0	a	a	a	a	a	a

A return code relating to the execution of the AUDIT macro is transferred in register R15.

X'aaaaaa'	Meaning
X'000000'	a) The requested action was accepted b) The requested action was ignored because AUDIT is already running (ACTION=ON) c) The requested action was ignored because AUDIT is not running (ACTION=DISC) d) The requested action was ignored because no AUDIT table has been allocated (ACTION=OFF) e) The requested action with SCOPE=FUNCT was ignored because AUDIT has already been activated for SCOPE=TASK f) The requested action was ignored because the hardware AUDIT is not available on this hardware.
X'000004'	Operand error. The generated data area contains illegal operand combinations, or the user or the program is not privileged for the action requested
X'000008'	The address of the data area is invalid, or the data area is not aligned on a word boundary, or an internal error occurred
X'00000C'	The address in the TABLE operand (table transfer area) is invalid or the area is read-only
X'000010'	Insufficient memory is available for either class 3 original tables, class 4 management areas, class 5 save tables or class 5 management areas. The AUDIT action cannot be executed
X'000014'	The task with the specified TID or TSN does not exist
X'000018'	The PCB addressed by SCOPE=FUNCT does not exist
X'01FFFF'	The UNIT or FUNCTION number specified in the standard header is invalid. Processing is aborted
X'03FFFF'	Invalid version number in the standard header
X'400020'	The table transfer area for the linkage AUDIT table (ACTION=GET) is too small. Execution of the function was incomplete
X'400024'	The function could not be executed because a)a processor-local linkage AUDIT is running (STATE=SYS) or b)a linkage AUDIT is running in the TPR processor state (STATE=PROC) or c)a processor-local SIH linkage AUDIT is running (SCOPE=SYSGLOBE) or d)a processor-local SIH and TPR linkage AUDIT is running (SCOPE=SIHGLOB)
X'400028'	The TU save table is not contained in the transfer buffer due to an internal AUDIT error.
X'40002C'	The task identified by means of its TID or TSN is terminating. The job was rejected.

X'aaaaaa'	Meaning
X'810030'	The macro call sequence AUDIT FCT=HWA,ACTION=ON/-OFF/-ON for an external task in TPR always produces this return code in the second ACTION=ON if the external task has not been executed in the meantime. This means that the second macro call with ACTION=ON must be repeated later because the macro call with ACTION=OFF is still in a queue.
X'810034'	Macro call AUDIT FCT=HWA/LNKA,STATE=U System administration has locked the hardware or linkage AUDIT for the active user ID using the /MODIFY-USER-ATTRIBUTES command.
X'810038'	Macro call AUDIT FCT=HWA/LNKA,STATE=U The user has locked the hardware or linkage AUDIT for the active task using the /MODIFY-TEST-OPTIONS command.
X'81003C'	Macro call AUDIT FCT=HWA/LNKA System administration has prohibited all AUDIT macro calls for the current session by specifying the parameter AUDALLOW=NO in the startup parameter service.
X'81FFFF'	The central linkage AUDIT management is currently locked (please wait and repeat the job at a later time).

BIND – Link and load load unit

Application areat: Linking and loading; see page 47
Macro type: Type S, MF format **2**: standard/C/D/L/E/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **BIND** macro links another load unit into the executing program.

Macro format and description of operands

```
BIND

,ALTLIB=*DBLOPT / NO / YES / list-poss(2): *TASKLIB / *BLSLIB##
[,AMODE@=adr / (r) / label]
[,AMODCHK = *DBLOPT / STD / ADVANCED]
,AUTOLNK=*DBLOPT / YES / NO / ALTLIB
,BRANCH=NO / YES
,CLOSE=*DBLOPT / ALL / NONE / ALT
,ERREXIT=*DBLOPT / adr / (r) / label
,IGNATTR=*DBLOPT / NONE / READ
,INTVERS=BLSP2 / SRV001 / SRV002 / SRV003 / SRV004 / SRV005 / SRV006
[,LDINFO=*DBLOPT / DEF / MAP / NONE / REF ]
 ⎧ LIBNAM@=adr / (r) / label    ⎫
,⎨ LIBNAM=*DBLOPT / datei / *   ⎬
 ⎩ LIBLINK=name                 ⎭
 ⎧ LNKCTX@=adr / (r) / label ⎫
,⎩ LNKCTX=*DBLOPT / name     ⎭
,LNKCTXS=*DBLOPT / ANY / OLD / NEW
,LOAD=YES / NO / ILESERVER
[,LOAD@=adr / (r) / label]
,MAP=*DBLOPT / NO / BOTH / (BOTH,nn) / nn / SYSOUT
[,MPID=adr / (r) / label]
[,MSG=*DBLOPT / INFORMATION / WARNING / ERROR / NONE]
[,NACOL=*DBLOPT / STD / ABORT]
```

BIND (continued)

```
,OVERLAY=NO / YES
 { PGMVER@=adr / (r) / label          }
,{ PGMVERS=*DBLOPT / *STD / version   }
,PROGMOD=*DBLOPT / ANY / 24
[,PURESOR= list-poss(3): USERSHARE / SYSSHARE / LNKCTX ]
[ ,PURESTY=*DBLOPT / STD / USER]
[,{ REFCTX@=adr / (r) / label                 }]
  { REFCTX=name / (name1,name2,...name200)    }
,REFCTX#=0 / n
 { REPFIL@=adr / (r) / label  }
,{ REPFILE=*DBLOPT / file     }
,REPSCOP=*DBLOPT / CONTEXT / UNIT
[,RESORD= list-poss(4): LNKCTX / USERSHARE / SYSSHARE / REFCTX ]
,RESTYP=*DBLOPT / STD / USER
,SHARE=*DBLOPT / SYSTEM / NONE / USER / GROUP / USER_GROUP / GLOBAL / ALL
,{ SYMBOL@=adr / (r) / label }, SYMBLAD=adr / (r) / label
 { SYMBOL= name / *ALL       }
,SYMTYP=ANY / CSECT / ENTRY / CSEN / MODULE
[,TSTOPT=*DBLOPT / NONE / AID]
[,{ UNIT@=adr / (r) / label }]
  { UNIT= name              }
[,UNRES=*DBLOPT / STD / DELAY / DELAYWARN / ABORT]
,USRMAPI = NONE / STD / ALL
[,USRMAP@ = adr / (r) / <label> ]
[,USRMAPL = integer 1..21474836479]
[,USRUNR@ = adr / (r) / label]
[,USRUNRL = integer 1..21474836479]
,USRUNRI = STD / DELAY / BOTH
[,{ VERS@=adr / (r) / label }]
  { VERS=version            }
,XPAND=PARAM / XRC / USRMAP / USRUNR
[,XRC=adr / (r) / label]
,XRCL=28 / 36
,MF=S / C / D / E / L / M [,PARAM=adr / (r)] ,PREFIX=P / p [,LABEL=name]
```

The operands are described in alphabetical order below.

ALTLIB=
Specifies whether alternate libraries are to be searched for the object defined by SYMBOL@ or SYMBOL. Alternate libraries are assigned using the file link name BLSLIBnn (00≤nn≤99) or `$BLSLBnn`. They are also used for the autolink function of DBL.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, ALTLIB=NO applies.

NO
Alternate libraries or tasklibs will not be searched.

YES
Alternate libraries will be searched.

***TASKLIB**
May be specified only if INTVERS=SRVxxx and xxx ≥ 002.
Tasklibs are searched in the following order:

1. The library which is assigned with the SET-TASKLIB command

2. The $userid.TASKLIB library

   or, in the event that this does not exist:

   The TASKLIB library under the default system ID (DEFLUID user ID)

***BLSLIB##**
May be specified only if INTVERS=SRVxxx and xxx ≥ 002.
Alternate libraries will be searched.

Notes

- The operand values *TASKLIB and *BLSLIB## can be specified as a list. The order of values in this list defines the order in which the relevant libraries will be searched.
- The specifications ALTLIB=YES and ALTLIB=BLSLIB## have the same meaning.
- The ALTLIB=YES and LIBNAM@/LIBNAM/LIBLINK operands may be specified together in the same macro. If, however, the main library specified by LIBNAM@/LIBNAM/LIBLINK is not present, DBL will abort the processing. ALTLIB=YES cannot serve as a substitute for a missing or invalid main library.

AMODCHK=
Determines whether additional checks of the addressing mode should be performed during loading (only in conjunction with INTVERS=SRVxxx and xxx ≥ 005). If the AMODCHK operand is not specified, the value set in the START-EXECUTABLE-PROGRAM or LOAD-EXECUTABLE-PROGRAM (or START-PROGRAM or LOAD-PROGRAM) load call will be used as the default value.

***DBLOPT**
The operand value is taken over from the last call of the MODIFY-DBL-DEFAULTS command. If no value has yet been defined with MODIFY-DBL-DEFAULTS for the operand involved, AMODCHK=STD applies.

STD
Only the checks compatible with BLSSERV < V2.5 are executed.

ADVANCED
The same checks as for AMODE-CHECK = *STD are performed.
During loading a check is also performed to see whether inconsistencies can occur while resolving external references because of the load unit's addressing mode.

AMODE@=
Specifies the address of a 1-byte long field into which the DBL enters the addressing mode for the load unit call. The field must be aligned on a word boundary.
The following are possible values for the addressing mode:

2	for AMODE 24
1	for AMODE 31
4	for AMODE 32

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address. May be specified only if MF=S or MF=L.

AUTOLNK=
Specifies whether the autolink function of DBL is to be turned on or off.

<u>*DBLOPT</u>
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, AUTOLNK=YES applies.

YES
The autolink function is to be turned on.

NO
The autolink function is to be turned off. External references are resolved only with the help of programs (private and shared) that are already loaded.

ALTLIB
The autolink function accesses only alternate libraries, which are assigned using the file link name BLSLIBnn (00≤nn≤99) or $BLSLBnn.

BRANCH=
Specifies whether the calling program is to be resumed immediately after the load unit has been loaded or whether the loaded load unit will be processed. The addressing mode is set by DBL.

NO
After the load unit has been loaded the instruction following the **BIND** macro in the calling program will be executed.

YES
After being loaded the load unit will be processed first. DBL determines the address using the symbol or module name specified with SYMBOL@ or SYMBOL.

CLOSE=
Specifies whether libraries used by DBL are to be closed or remain open when processing of the DBL macro has been completed. This refers to all libraries which DBL searches for modules. The operand can be used to speed processing when DBL is called a number of times with the same library.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, CLOSE=ALL applies.

ALL
All libraries used will be closed.

NONE
All libraries used will remain open and can be used for a further DBL call.

ALT
All alternate libraries used will be closed. Only the main library specified by the LIBNAM@, LIBNAM or LIBLINK operand is to remain open.

ERREXIT=
Specifies the address of a 4-byte field.
If the UNRES operand is specified with the value STD/DELAY/DELAYWARN the address that is to be assigned to unresolved external references should be entered in this field.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, unresolved external references are assigned the address X'FFFFFFFF'.

addr
Address of an auxiliary field which contains the field address searched for. May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic address of the field. May be specified only if MF=S or MF=L.

IGNATTR=
Specifies which CSECT attributes are to be ignored during loading.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, IGNATTR=NONE applies.

NONE
All CSECT attributes are taken into account during loading.

READ
The CSECT attribute READ-ONLY is ignored during loading. The CSECT is loaded into a readable/writable main memory page. This makes it possible to set breakpoints when debugging using AID, for example.

INTVERS=
The operand specifies the version of the BIND macro interface.

BLSP2
Default. Corresponds to macro version 5.

SRV001
Corresponds to macro version 6. This version is supported by BLSSERV as of V2.2.

SRV002
Corresponds to macro version 7. This version is supported by BLSSERV as of V2.3A.

SRV003
Corresponds to macro version 8. This version is supported by BLSSERV as of V2.3B.

SRV004
Corresponds to macro version 9. This version is supported by BLSSERV as of V2.4A.

SRV005
Corresponds to macro version 10. This version is supported by BLSSERV as of V2.5A.

SRV006
Corresponds to macro version 11. This version is supported by BLSSERV as of V2.6A.

LABEL=name
May be specified only if MF=M.
Name of the structure, i.e. the DSECT which describes the operands of the **BIND** macro. The operand is mandatory if there is no valid USING statement for the definition of the base address register for the DSECT of the parameter list. The LABEL operand must be specified in conjunction with the PARAM operand. Both operands are used to produce a valid USING statement.

The following may be specified for “name”:
- The name specified in the name field of a preceding macro `name BIND MF=D`.
- The name “xPBBNDS” if no “name” has already been specified, where “x” is the value of the PREFIX operand of a preceding macro `BIND MF=D, PREFIX=x`. The default value for “x” is “P”.
- The name of the longer DSECT containing the parameter list of the **BIND** macro if the macro `BIND MF=C` was specified earlier.

LDINFO=
Defines the load information for the load unit.
If the LDINFO operand is not specified, the value set with the START-EXECUTABLE-PROGRAM or LOAD-EXECUTABLE-PROGRAM (or START-PROGRAM or LOAD-PROGRAM) load call is used as the default value.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, the value that follows *DBLOPT in the syntax definition applies.

DEF
An external symbol dictionary containing the program definitions of all modules in the load unit will be loaded.
Program definitions are control sections (CSECTs), entry points (ENTRYs), COMMONs, dummy sections (DSECTs), external dummy sections (XDSEC-Ds) and module names.

MAP
Only an external symbol dictionary which is required for building the DBL map will be loaded *temporarily*. The external symbol dictionary will be unloaded as soon as the DBL map is built.

NONE
No external symbol dictionary will be loaded.

REF
An external symbol dictionary will be loaded containing the resolved references of all modules in the load unit in addition to the program definitions. References are external references (EXTRNs), V-type constants, weak external references (WXTRNs) and external dummy sections (XDSEC-Rs).

LIBLINK=name
File link name of the main library. The name may be up to 8 characters long.

LIBNAM@=
Specifies the main library in which the search for the object defined with SYMBOL@ or SYMBOL is to take place. The main library is defined through the address of a field that contains the file name of the library. The EAM object module file is defined using the file name “*”. It is not possible to specify a file link name for the EAM object module file.
The main library is searched *before* the alternate libraries (see page 264). It is also used for the autolink function of DBL. If the LIBLINK operand is used, any LIBNAM@ entry is ignored.

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic ield address. May be specified only if MF=S or MF=L.

LIBNAM=
File name of the main library. May be specified only if MF=S or MF=L.
If the LIBLINK operand is used, any LIBNAM entry is ignored.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, the library with the file link name BLSLIB is searched.

file
Explicit specification of the file name of the main library. The file name may be up to 54 characters long.

*****
Specifies the EAM object module file as the main library.

LNKCTX@=
Specifies the address of a field containing the name of the link context. The name must begin with a letter.

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address. May be specified only if MF=S or MF=L.

LNKCTX=
May be specified only if MF=S or MF=L.

name
Explicit specification of the name for the link context. The name may be up to 32 characters long and must begin with a letter.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, the link context “LOCAL#DEFAULT” is used.

LNKCTXS=
Specifies whether or not the new link context defined via LNKCTX@ or LNKCTX may be located within the existing user contexts.

<u>*DBLOPT</u>
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, LNKCTXS=ANY applies.

ANY
If a user context with the name specified by means of the LNKCTX@ or LNKCTX operand is present in the user context, the existing context will be used. If no context with the same name is present a new context will be generated.

OLD
The specified link context must already be located within the existing user context. A new context will not be generated.

NEW
The specified link context must not be located within the existing user context. A new context will be generated.

LOAD=
Specifies whether a symbol with the symbol name SYMBOL@ or SYMBOL is to be loaded or whether loading of an ILE server module is required. This will depend on whether the DBL takes ILE type symbols into account when searching for the primary input.

<u>YES</u>
The symbol will be loaded if it is not already loaded. ILE symbols will be included.

NO
The symbol will not be loaded. If it is already loaded the load address will be passed by DBL. ILE symbols will be included.
The **BIND**...,LOAD=NO call should be replaced by the **VSVI1**...,SELECT=BYNAME call for performance reasons.

ILESERVER
The symbol to be loaded is the entry point of an ILE server module. ILE symbols will therefore be ignored.

LOAD@=

Specifies the address of an area below 16 Mb in class 6 memory at which the first module is to be loaded. If other modules are present they will be loaded into a free area after the first module.

If the address is located above 16 Mb in class 6 memory the LOAD@ operand is ignored and the OVERLAY operand set to NO. Loading is not performed if:

– the address is not aligned on a doubleword boundary or an invalid address in class 6 memory was specified,

– the user has defined multiple contexts in class 6 memory,

– the area of class 6 memory in which the first module is to be loaded is already occupied and the OVERLAY=NO operand was specified,

– an LLM was loaded either as the first module or using autolink,

– the MPID operand was also specified.

If the LOAD@ operand is not specified DBL determines the load address. If a load address was specified when storing an LLM, the LLM is loaded starting at this address if this is possible.

addr

Address of a field which contains the area address. May be specified only if MF=M.

(r)

r = register containing the area address searched for. May be specified only if MF=M.

label

Area address. The address can be specified as a symbolic address or as a constant (X'...'). May be specified only if MF=S or MF=L.

MAP=

Specifies whether a DBL map is to be output or not, and if so, indicates the output destination.

***DBLOPT**

The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, MAP=NO applies.

NO

No DBL map is to be output.

BOTH

The output destination is the SYSOUT and SYSLST00 system files.

(BOTH,nn)

The output destination is the SYSOUT system file and a SYSLSTnn system file (where 00≤nn≤99).

nn
The output destination is a system file from the set SYSLST00 to SYSLST99 whose number must be specified here.
The number must given in 2-digit form (00 for 0, etc.).

SYSOUT
The output destination is the SYSOUT system file.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM and PREFIX, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form or D form of the macro (see section “S-type macros” on page 29).

MPID=
Address of a 4 byte field which contains the identifier of the memory pool into which the load unit is loaded.
This short identifier is made available to the user by means of the **ENAMP** macro.
The operand must not be specified:
- if the LOAD@ and OVERLAY operands are specified,
- if an LLM containing user-defined slices is loaded.

The user is responsible for management of the memory pool. DBL passes information concerning the loaded load unit in the memory pool only to the user who issued the **BIND** macro call.
Multiple load units can be loaded into a memory pool within a user context. Shared code that is to be made available as such in memory pools must be loaded using the **ASHARE** macro.

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address. May be specified only if MF=S or MF=L.

MSG=
Specifies the lowest message class; messages at and above this level will be output.
If the MSG operand is not specified, the value set in the START-EXECUTABLE-PROGRAM or LOAD-EXECUTABLE-PROGRAM (or START-PROGRAM or LOAD-PROGRAM) load call will be used as the default value.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, the value that follows *DBLOPT in the syntax definition applies.

INFORMATION
All classes of message will be output.

WARNING
Only messages of the WARNING and ERROR classes will be output. Messages of the INFORMATION message class will not be output.

ERROR
Only messages of the ERROR class will be output.

NONE
No messages will be output.

NACOL=
Defines how name conflicts affecting symbols with identical names are to be handled. Name conflicts are detected only if the symbols are *not* masked.
If the NACOL operand is not specified the value set in the START-PROGRAM or LOAD-PROGRAM load call is used as the default value.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, NACOL=STD applies.

STD
Name conflicts between nonmasked symbols will be indicated by warning messages. The module containing the symbol with the same name will be loaded. The new occurrence of the symbol will be masked, i.e. it will no longer be used for resolving external references.

ABORT
Loading of the current load unit will be aborted if a name conflict between nonmasked symbols is detected.

OVERLAY=
Specifies whether the modules of the load unit may overlay the module located at the address specified with LOAD@. This operand requires the LOAD@ operand to be specified and is incompatible with the MPID operand.

NO
The modules of the load unit must not overlay the module located at the address specified with LOAD@.

YES
The modules of the load unit may overlay the LOAD@ module. The first module of the load unit is loaded at the address specified with LOAD@. If other modules are present they will be loaded contiguously after the first. An exception are areas locked by means of the CSECT attributes READ-ONLY and PAGEABLE. The user must ensure that the overlaid area is no longer needed later.
All modules overlaid by modules of the load unit are unloaded by DBL. The used memory contents are not deleted before the new load operation, however.
COMMONs are loaded independently of the specified load address.

PGMVER@=
Specifies the address of a field containing the program version. If this program version is already loaded, a link to it is created. If this program version is not loaded, the new load unit receives the specified version.

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address. May be specified only if MF=S or MF=L.

PGMVERS=
Specifies the program version.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, PGMVERS=*STD applies.

***STD**
The load unit resulting from the load call receives the version of the loaded library element as the program version. If the symbol specified in the load call is already loaded, a search is carried out for the program version that was set using the SELECT-PROGRAM-VERSION command. If no program version has been set, DBL uses the first symbol found.

version
may be up to 24 characters long.

PROGMOD=
Specifies in which part of the address space (above or below 16 Mb) the modules of the load unit are to be loaded.

<u>*DBLOPT</u>
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, PROGMOD=ANY applies.

ANY
The modules of the load unit can be loaded above or below 16 Mbytes.

24
The entire load unit will be loaded below 16 Mb.
The program will be executed in 24-bit addressing mode.
External references will be interpreted as 24-bit addresses.

PURESOR
Specifies the user-defined search sequence for the resolution of external references in PUBLIC parts of LLMs if PURESTY=USER was specified (only in conjunction with INTVERS=SRVSRVxxx and xxx ≥ 001).

list-poss(3): USERSHARE / SYSSHARE / LNKCTX
The search sequence is defined by the sequence of the keywords within the list. The keywords have the following meaning:

USERSHARE	for shared code of the user
SYSSHARE	for shared code of the system
LNKCTX	for link context

If PURESOR=(SYSSHARE, LNKCTX, USERSHARE) is specified, for example, when resolving external references in PUBLIC parts the search is carried out in the following sequence:

1. in the shared code of the system
2. in the link context
3. in the shared code of the user

Notes

– Each keyword can only occur once in the list.

– Keywords that are not specified in the list are appended to the end of the list internally by DBL in accordance with the predefined sequence (USERSHARE, SYSSHARE, LNKCTX). For example, PURESOR=(SYSSHARE) is handled in the same way as PURESOR=(SYSSHARE, USERSHARE, LNKCTX).

– The PURESTY and PURESOR operands have no influence on **which** contexts are searched. They only specify the sequence. **Whether** shared code is searched must be specified with the SHARE operand. DBL does not carry out any consistency checks on these operands. If PURESOR=(LNKCTX, SYSSHARE), SHARE=NONE is specified, for example, the shared code of the system is not used for the resolution of external references although the corresponding keyword is specified in the list of the search sequence.

PURESTY=
Specifies the search strategy for the resolution of external references in PUBLIC parts of LLMs (only in conjunction with INTVERS=SRVxxx and xxx ≥ 001).

<u>*DBLOPT</u>
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, PURESTY=STD applies.

STD
The search sequence predefined by DBL applies:

1. shared code of the user
2. shared code of the system
3. link context

USER
The search sequence is specified by the user with the PURESOR operand.

REFCTX@=
Specifies the address of a field containing a name list of reference contexts which will be searched in order to resolve external references.
Up to 200 names of reference contexts can be entered in the list. A name must begin with a letter. The contexts specified in the list must be present. The reference contexts are searched in the order in which they appear in the list. The number of reference contexts can be defined using the REFCTX# operand. A link context defined using LNKCTX@ or LNKCTX cannot be used as a reference context.

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address. May be specified only if MF=S or MF=L.

REFCTX=name / (name1,...,name200)
May be specified only if MF=S or MF=L.
Explicit specification of a list containing the names of reference contexts. Each name may consist of up to 32 characters. Up to 200 names may be entered in the list. The names must not begin with a dollar sign ($).

REFCTX#=0 / n
Specifies the number of reference contexts in the name list defined with REFCTX@ or REFCTX.
0≤n≤200; Default value: 0

REPFIL@=
Specifies the address of a field which contains the name of a REP file.
This enables the user to apply REP records to the modules of a load unit. The REP records must have the standard record format for processing by the RMS utility routine (see the “Utility Routines” manual [27]). If there is an error during processing of REP records, a warning message is output and the errored REP record is skipped. REP processing then resumes.

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address. May be specified only if MF=S or MF=L.

REPFILE=
May be specified only if MF=S or MF=L.
Specifies the name of the REP file.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, no REP file is used.

file
Explicit specification of the name of the REP file. The name may be up to 54 characters long.

REPSCOP=
Specifies whether REP processing is to be performed for all modules in the context or only for modules in the current load unit.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, REPSCOP=CONTXT applies.

CONTEXT
REP processing is to be performed for all modules in the context.

UNIT
REP processing is to be performed only for modules in the current load unit. All other modules in the context will be skipped.

RESORD=
Specifies the user-defined search sequence for the resolution of external references if RESTYP=USER was specified (only in conjunction with INTVERS=SRVxxx and xxx ≥ 001).

list-poss(4): LNKCTX / USERSHARE / SYSSHARE / REFCTX
The search sequence is defined by the sequence of the keywords within the list. The keywords have the following meaning:

LNKCTX	for link context
USERSHARE	for shared code of the user
SYSSHARE	for shared code of the system
REFCTX	for reference context

If RESORD=(REFCTX, USERSHARE, SYSSHARE, LNKCTX) is specified, for example, the search is carried out in the following order when resolving external references:

1. in the reference context
2. in the shared code of the user
3. in the shared code of the system
4. in the link context

Notes

- Each keyword can appear in the list only once.
- Keywords that are not specified in the list are appended to the end of the list internally by DBL in accordance with the predefined sequence (LNKCTX, USERSHARE, SYSSHARE, REFCTX). For example, RESORD=(REFCTX, USERSHARE) is handled in the same way as RESORD=(REFCTX, USERSHARE, LNKCTX, SYSSHARE).

– The RESTYP and RESORD operands have no influence on **which** contexts are searched. They only specify the sequence. **Whether** shared code or reference context is searched must be specified with the SHARE or REFCTX operand (or REFCTX@). DBL does not carry out any consistency checks on these operands. If RESORD=(LNKCTX, SYSSHARE), SHARE=NONE is specified, for example, the shared code of the system is not used for the resolution of external references although the corresponding keyword is specified in the list of the search sequence.

RESTYP=
Specifies the search strategy for the resolution of external references (only in conjunction with INTVERS=SRV001/SRV002).

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, RESTYP=STD applies.

STD
The search sequence predefined by DBL applies:

1. link context
2. shared code of the user
3. shared code of the system
4. reference context(s)

USER
The search sequence is specified by the user with the RESORD operand.

SHARE=
Specifies what part of the shared code is to be included in the search for the symbol specified with SYMBOL@ or SYMBOL and for resolving external references. This also applies to the autolink function of DBL if AUTOLNK=YES is specified. If the symbol involved is located in a common memory pool, DBL returns the load address, links the user task with the common memory pool and terminates the load procedure. If the symbol involved is located in a nonprivileged subsystem (see “Subsystem Management” manual [12]), DBL returns the load address, establishes a connection to the subsystem and terminates the load procedure. If BRANCH=YES is specified, a branch is then made to the address of the symbol found.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, SHARE=SYSTEM applies.

SYSTEM
Only the shared code of the system (in class 3/4/5 memory) is included in the search.

NONE
No shared code is to be included in the search. DBL causes a private copy of the program to be loaded.

USER
Only the user's shared code in common memory pools is to be included in the search, irrespective of the scope of the common memory pool.

GROUP
Only shared code in common memory pools with the scope GROUP is to be included in the search.

USER_GROUP
Only shared code in common memory pools with the scope USER_GROUP is included in the search.

GLOBAL
Only shared code in common memory pools with the scope GLOBAL is included in the search.

ALL
The shared code of both the system and the user is included in the search.

SYMBLAD=
Specifies the address of a 4-byte field, aligned on a word boundary. In this field DBL enters the address at which the program run has to be continued when the load unit is referenced.

This address depends on the operand SYMTYP:

- If SYMTYP=MODULE or SYMTYP=ANY is specified and a module with the name that was specified with the SYMBOL (or SYMBOL@) operand is loaded, then the address is the start address of the module (LLM or OM). Details for computing the start address of LLMs see manual “BINDER” [5]. If the start address of the LLM is an external name, the address of that CSECT or ENTRY is returned.
- In all other cases (if SYMTYP=CSECT, ENTRY or CSEN is specified or SYMTYP=MODULE or SYMTYP=ANY is specified and a module with another name than specified with the SYMBOL (or SYMBOL@) operand was loaded) the returned address is the address of the CSECT or ENTRY with the name specified with the SYMBOL (or SYMBOL@) operand.

The addressing mode to be used for the load unit call is indicated as following:

- on a server with /390 architecture in the leftmost bit of the field specified with the SYMBLAD operand,
- on other BS2000 servers in the field AMODE@ if specified.

In the following cases loading is aborted:

- SYMBLAD is not specified
- the field is not aligned on a word boundary
- the according memory area has no write access or is not allocated.

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Field address. The address can be specified as a symbolic address or as a constant (X'...'). May be specified only if MF=S or MF=L.

SYMBOL@=
Specifies the address of a field containing the name of an object. DBL uses this name to determine which module of the load unit is to be loaded first. The name can refer to the following objects:

- control section (CSECT),
- entry point (ENTRY),
- object module (OM) (element name),
- link and load module (LLM) (element name).

The type of the object is defined using the SYMTYP operand.

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address. May be specified only if MF=S or MF=L.

i The field which contains the name must be 32 characters long. If the name is shorter, it must be padded with blanks.

SYMBOL=
May be specified only if MF=S or MF=L.
Specifies the name of a control section (CSECT), an entry point (ENTRY), an object module (OM) or a linking loader module (LLM). DBL uses this name to determine which module of the load unit is to be loaded first.

name
Explicit specification of the object name. The name may be up to 32 characters long.

An asterisk (*) can be specified as the last character of the name. This represents any character string. In this case all library elements whose name corresponds to the specified pattern are loaded from the main library (LIBNAM/LIBNAM@/LIBLINK operand) into **one** list name unit.

i If multiple asterisks (*) are specified in the name, all characters after the first asterisk are ignored.

***ALL**
All library elements of the main library (LIBNAM/LIBNAM@/LIBLINK operand) are loaded into **one** list name unit.

SYMTYP=
Specifies the type of object defined with the name SYMBOL@ or SYMBOL and defines the search sequence for the object. A symbol (CSECT or ENTRY) or module (OM or LLM) can be defined as the type of object.
If the object is a symbol the name SYMBOL@ or SYMBOL designates a symbol name and can be:

- the name of a nonmasked CSECT or ENTRY entry in a program library (type R or type L),
- the name of a nonmasked CSECT or ENTRY entry in an object module library (OML) or in the EAM object module file (OMF).

If the object is a module the name SYMBOL@ or SYMBOL designates a module name and can be:

- the name of a library element (type R or type L) in a program library,
- the name of a library element in an object module library (OML).

ANY
All symbol tables are included in the search. The search is performed in the following sequence, as defined by DBL:

1. LLMs with the module name SYMBOL@ or SYMBOL
2. OMs with the module name SYMBOL@ or SYMBOL
3. Symbols with the symbol name SYMBOL@ or SYMBOL in an LLM. DBL searches for CSECTs first. If no CSECT is found, it searches for ENTRYs.
4. Symbols with the symbol name SYMBOL@ or SYMBOL in an OM. DBL searches for CSECTs first. If no CSECT is found, it searches for ENTRYs.

CSECT
Only control sections (CSECTs) with the symbol name SYMBOL@ or SYMBOL are searched for.

ENTRY
Only entry points (ENTRYs) with the symbol name SYMBOL@ or SYMBOL are searched for.

CSEN
CSECTs *and* ENTRYs with the symbol name SYMBOL@ or SYMBOL are searched for. DBL searches for CSECTs first. If no CSECT is found, it searches for ENTRYs.

MODULE
Only modules with the module name SYMBOL@ or SYMBOL are searched for.

i A search for any already loaded CSECT or ENTRY names does not take place. In this way, repeated BIND calls can lead to multiple loading, which can result in BLS0339 messages.

TSTOPT=
Specifies whether symbolic addresses in the source program may be used for debugging with AID. Only programs for which test and diagnostic information (LSD) has been generated during compilation may be debugged using symbolic addresses. For this purpose certain compiler options must be set when compiling the source program (refer to user guide for the relevant compiler or assembler). If the TSTOPT operand is not specified, the value set with the START-EXECUTABLE-PROGRAM or LOAD-EXECUTABLE-PROGRAM (or START-PROGRAM or LOAD-PROGRAM) load call is used as the default value.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, TSTOPT=NONE applies.

NONE
Test and diagnostic information (LSD) is not taken into account.

AID
Permits symbolic addresses from the source program to be used when debugging the program with AID (see the “AID” manual [3]). This entry is valid only if LDINFO=DEF or LDINFO=REF is specified at the same time.

UNIT@=
Specifies the address of a field containing the name of the load unit. The name can be used in subsequent **UNBIND** macros.
If the operand is not given, the name defined via SYMBOL@ or SYMBOL will be used.

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address. May be specified only if MF=S or MF=L.

UNIT=name
May be specified only if MF=S or MF=L.
Explicit specification of the name of the load unit. The name may be up to 32 characters long.

UNRES=
Specifies how unresolved external references are to be handled. All unresolved external references will be output to the SYSOUT system file, with external dummy sections (XDSECs-R) being listed separately.
If the UNRES operand is not specified the value set in the START-EXECUTABLE-PROGRAM or LOAD-EXECUTABLE-PROGRAM (or START-PROGRAM or LOAD-PROGRAM) load call will be used as the default value.

***DBLOPT**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, the value that follows *DBLOPT in the syntax definition applies.

STD
Unresolved external references (except for external dummy sections (XDSECs-R)) will be given an address which is specified in the ERREXIT operand.

DELAY
Unresolved external references will be resolved at a later time. This operand is valid only in combination with LDINFO=REF.
DBL stores the unresolved external references in the link context. When the next load unit is loaded in the context DBL attempts to resolve the stored external references with CSECTs and ENTRYs from this load unit at the end of the loading operation. This process is repeated each time a new load unit is loaded for as long as the context exists. External dummy sections (XDSEC-Rs) cannot be stored.
When stored in the context, the unresolved external references are given a (provisional) address which is specified in the ERROR-EXIT operand.

DELAYWARN
Specified only in conjunction with INTVERS=SRVxxx, where xxx ≥ 004.
The behavior is like UNRES=DELAY. In addition, when unresolved external references occur, a corresponding return code is issued.

UNRES=DELAYWARN is also a prerequisite for USRUNRI=DELAY.

ABORT
Unresolved external references are not allowed. Loading of the current load unit will be aborted.

USRMAP@=
Address of a user-defined data area to which information of the DBL map which is defined with the USRMAPI operand (only in conjunction with INTVERS=SRVxxx and xxx ≥ 005) is to be output. The data area must be aligned on word boundary. The length of the area must be transferred with the USRMAPL operand. You can also obtain the layout of the information with
`BIND MF=D,XPAND=USRMAP,INTVERS=SRV005`.

This specification is ignored if USRMAPI=NONE.

addr
Address of an auxiliary field which contains the address of the data area.
May be specified only if MF=M.

(r)
r = register containing the address of the data area. May be specified only if MF=M.

label
Symbolic address of the data area. May be specified only if MF=S or MF=L.

USRMAPI=
Determines which part of the DBL map is to be output to a data area whose address the user transfers with the USRMAP@ operand (only in conjunction with INTVERS=SRVxxx and xxx ≥ 005).

NONE
The DBL map is not output to a user-defined data area.

STD
The header of the DBL map and the information on load unit and modules are output to a user-defined data area, but no information on CSECTs and ENTRYs.

ALL
The entire DBL map (as with SYSOUT) is output to a user-defined data area.

USRMAPL=integer 1..21474836479
Length of the user-defined data area whose address is transferred with the USRMAP@ operand (only in conjunction with INTVERS=SRVxxx and xxx ≥ 005). The length must be specified in bytes. It must (for the minimum information) be at least 248 bytes for INTVERS=SRV005 and 336 bytes for INTVERS=SRV006.

This specification is ignored if USRMAPI=NONE.

USRUNR@=
Address of a user-defined data area to which a list of the unresolved external references is to be output (only in conjunction with INTVERS=SRVxxx and xxx ≥ 005). The data area must be aligned on word boundary. The length of the area must be transferred with the USRUNRL operand.

The information output mainly consists of a list header, the length of the information output and the number of unresolved external references, and a list of records, each of which define one of these external references.
The layout of the area can be generated with `BIND MF=D,XPAND=USRUNR,INTVERS=SRVxxx` where xxx ≥ 005. The layout depends on the version (`INTVERS` operand).

addr
Address of an auxiliary field which contains the address of the data area.
May be specified only if MF=M.

(r)
r = register containing the address of the data area. May be specified only if MF=M.

label
Symbolic field address of the data area. May be specified only if MF=S or MF=L.

USRUNRI=
Determines which unresolved external references are to be output to the data area whose address the user transfers with the USRUNR@ operand (only in conjunction with INTVERS=SRVxxx und xxx ≥ 006).

The specification is ignored if USRUNR@ is not specified.

STD
The external references from the current load operation which remain unresolved are output to the user-defined data area.

DELAY
The external references from earlier load operations (with UNRES=DELAY) which remained unresolved are output to the user-defined data area.

When USRUNRI=DELAY, UNRES=DELAYWARN must also be specified.

BOTH
Both types of unresolved external references are output to the user-defined data area.

When USRUNRI=BOTH, UNRES=DELAYWARN must also be specified.

USRUNRL=integer 1..21474836479
Length of the user-defined data area whose address is transferred with the USRUNR@ operand (only in conjunction with INTVERS=SRVxxx and xxx ≥ 005). The length must be specified in bytes. It must (for the output header) be at least16 bytes for INTVERS=SRV005 and 20 bytes for INTVERS=SRV006.

This specification is ignored if USRUNR@ is not specified.

VERS@=
Specifies the address of a field containing the element version of the element defined by SYMBOL@ or SYMBOL. If SYMTYP=ANY is specified, the VERS@ operand is taken into account only if the object name (SYMBOL@ or SYMBOL) refers to a module in a program library (type R or type L). If a CSECT or ENTRY name is specified, the VERS@ operand is ignored. If the operand is not specified, the default value for the highest element version in program libraries is used (see the “LMS” manual [29]).

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address. May be specified only if MF=S or MF=L.

VERS=version
May be specified only if MF=S or MF=L.
Explicit specification of the element version. The name may be up to 24 characters long.

XPAND=
Specified only with MF=D.
Determines the layout of the data area to be generated.

PARAM
Generates the layout of the parameter list for calling the BIND macro.

XRC
Generates the layout for the extended return code.

USRMAP
Specified only in conjunction with INTVERS=SRVxxx and xxx ≥ 005.
Generates the layout for the data area to which the DBL map can be output (see USRMAPI, USRMAP@ and USRMAPL operands).

USRUNR
Specified only in conjunction with INTVERS=SRVxxx and xxx ≥ 005.
Generates the layout for the data area to which a list of the unresolved external references can be output (see USRUNR@, USRUNRI and USRUNRL operands).

XRC=
Specifies the address of a field containing the **extended return code**.
The address must be aligned on a word boundary. The field has the following format:

Byte	Length	Content
0- 6	7	Message code of the last message output during processing of the BIND macro
7	1	blank
8-11	4	DMS error code
12-23	12	PLAM error code
24-27	4	Error code for other types of error that may occur when searching in libraries
28	1	ILE flag and 7 reserved bits
29-31	3	reserved
32	4	Address of the ILE server module

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address. May be specified only if MF=S or MF=L.

XRCL=
Defines the length of the field specified in XRC. Depending on the length set, DBL transfers all or only some of the possible information.

28
The XRC field is 28 bytes long. No information about ILEs will be transferred.

36
The XRC field is 36 bytes long. All information, including ILE information, will be transferred.

Notes on the macro call

- Modules or control sections (CSECTs) which have already been loaded once in the link context are not loaded a second time by DBL. With BRANCH=YES DBL branches to the address of the symbol (CSECT or ENTRY) that was loaded. With BRANCH=NO processing is continued with the next instruction in the calling program.
- CSECTs or COMMONs which have names consisting of blanks are given the default name “%CSECT” or “%COM”.
- If an LLM containing no relocation information cannot be loaded at a suitable address an error condition is produced.
- When loading the load unit DBL attempts to resolve unresolved external references from a previous load unit which were stored in the link context (UNRES=DELAY operand) for the current load unit.
- The OPEN mode for the program libraries and elements is determined by the value defined with the system parameter BLSOPENX (see the “Introduction to System Administration” manual [10]).
- The ALTLIB=YES and LIBNAM@/LIBNAM/LIBLINK operands may be specified together in the same macro. If, however, the main library specified by the LIBNAM@/LIBNAM/LIBLINK operand is not present, DBL aborts the processing. Specifying ALTLIB=YES is no substitute for a missing or invalid main library.
- If the specified program version has not yet been loaded but a program with this name already exists in the link context (see CONTEXT operand), loading is rejected because of the name conflict.
- When a REP file (REPFILE, REPFIL@) is specified, DBL can also process an associated NOREF file provided the name convention for BS2000-REP files is adhered to: either the NOREF file name includes the element “SYSNRF” (instead of SYSREP), or the NOREF file has the same name as the main library with the extension “.NOREF”. NOREF files must have the same format in the user environment as in the system environment.
- When `BIND MF=D,XPAND=USRMAP` is specified, the same prefix must be specified as for `BIND MF=D,XPAND=PARAM`.
- If the user-defined output area for the DBL map (USRMAPI, USRMAP@ and USRMAPL operands) is too small to contain all the information, a return code (X’08010129’) is entered in the map header. Output is aborted, but the load operation is continued.
- The list of unresolved external references or external references from earlier load operations (with UNRES=DELAY) which remained unresolved (if there is one) is output to the area defined with USRUNR@, USRUNRI and USRUNRL. This output is independent of the UNRES operand. If the area that is to contain all the information is

too small, a return code (X’0801012D’) is entered in the map header. Output is aborted, but the load operation is continued. If all external references have been resolved, 0 is entered in the map header.

– When `SYMBOL=*ALL` or `SYMBOL=name` is specified and the last character of `name` being the wildcard symbol “*”, the result of the load process is a socalled list name unit.
For details, see the „BLSSERV“ manual [4].

Format of the name structure in the parameter list of the BIND macro

DBL manages the various names defined in the **BIND** macro with the aid of the **BIND** macro parameter list. For each name the parameter list contains a pointer which points to the address of the associated name field (see figure on the next page). Except for SYMBOL@ all names are optional. If a name is not defined in the macro the macro sets the associated pointer to the address value X'FFFFFFFF'.

A name can also be explicitly specified in the macro. In this case the names are entered in the parameter list.

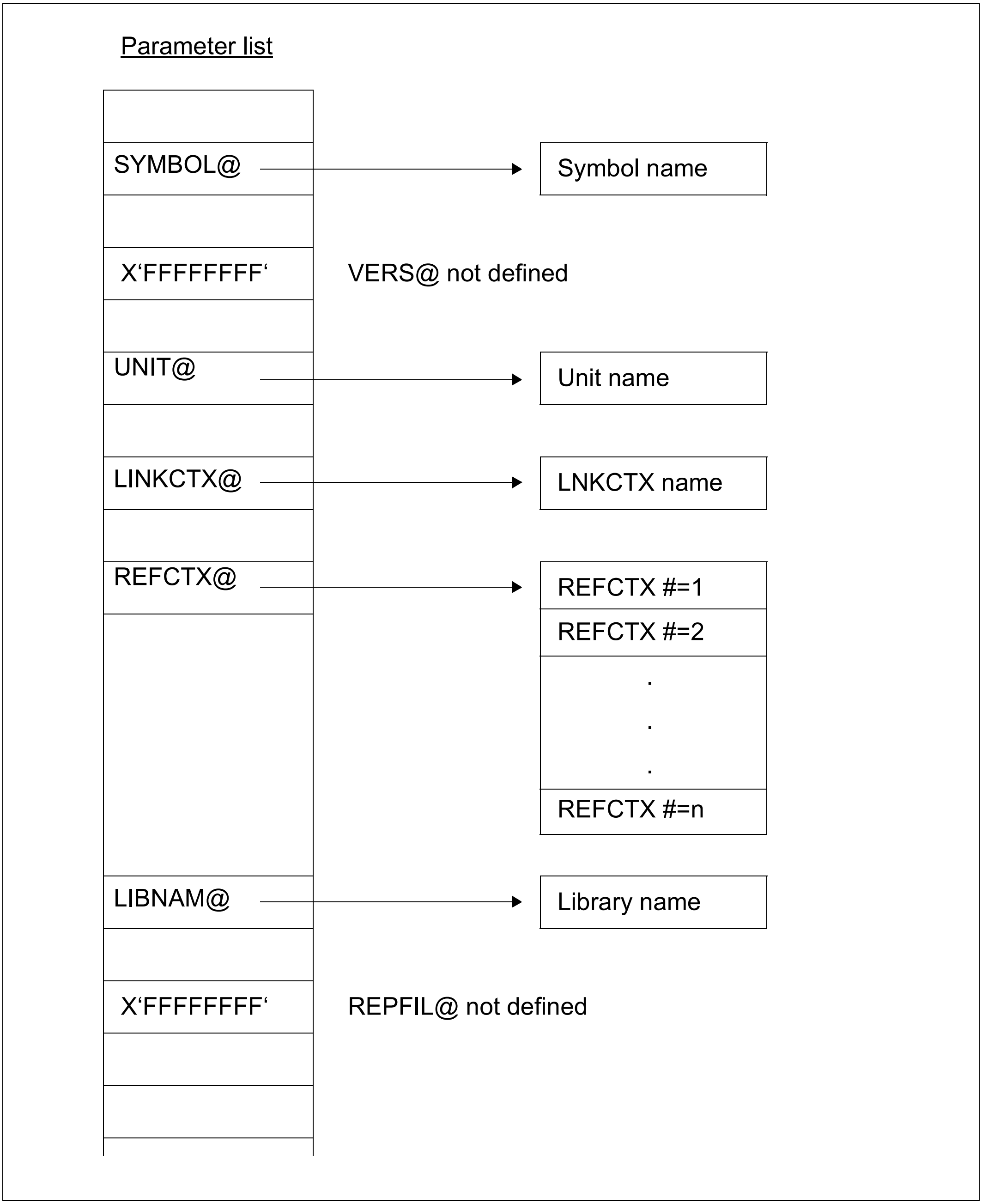


Figure 23: Name structure in the parameter list of the BIND macro

Search strategy

DBL searches in various containers for the object defined using SYMBOL@ and SYMTYP. If a suitable object is found in a container it is inserted in the load unit, the load unit is loaded and the start address passed to the user via the SYMBLAD field.

The search operation is performed in the following stages:

1. Search in the link context. The reference context is not searched.
2. Search in the user's shared code that was loaded into a common memory pool by means of the **ASHARE** macro of DBL. The search can be restricted to memory pools with a particular scope or suppressed in its entirety (SHARE-SCOPE operand).
3. Search in the shared code in the system address space in which nonprivileged subsystems are loaded (see the “Subsystem Management” manual [12]). The user can suppress searching in nonprivileged subsystems by specifying SHARE-SCOPE ≠ SYSTEM in the load call.
4. Search libraries which the user specified with the LIBNAM@ or LIBNAM or LIBLINK operand in the load call.
5. Search alternate (system) libraries assigned using the file link name BLSLIBnn (where 00≤nn≤99) or $BLSLBnn. The libraries are searched in ascending numerical order (by “nn”). The alternate system libraries $BLSLB00..49 are searched first, followed by the alternate libraries BLSLIB00..99, and finally, the remaining alternate system libraries $BLSLB50..99.

   The system and/or user tasklibs can be searched, depending on the operand ALTLIB.

   The user can suppress searching in alternate libraries by specifying the ALTLIB=NO operand in the load call.

Note

For list name units, no names in the link context or shared code are searched for as these names are element names and not symbol names.

For details, see the „BLSSERV“ manual [4].

Return information and error flags

The start address of the load unit is transferred to the field specified with the SYMBLAD operand. The high-order bit of the transferred start address indicates which addressing mode has to be set (bit = 1 for 31-bit addressing, bit = 0 for 24-bit addressing).

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the BIND macro is transferred in the standard header (cc=Subcode2,bb=Subcode1,aaaa=Maincode).

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Macro executed normally
X'0C'	X'01'	X'0018'	A reserved field of the parameter list is not prefilled with zeros
X'0C'	X'01'	X'0100'	Illegal combination of parameters in the parameter list. Illegal combinations may, for example, include: – OVERLAY=YES specified without LOAD@ – LDINFO=NONE specified together with TEST-OPTIONS in the LOAD-EXECUTABLE-PROGRAM or LOAD-PROGRAM command – RESTYP=USER specified without RESORD – USRMAPI=STD/ALL specified without USRMAP@ or USRMAPL
X'0C'	X'01'	X'0104'	The macro call is invalid because it was issued under a user ID that is permitted for maintenance only
X'0C'	X'40'	X'0114'	LNKCTXS=OLD was given and the specified context is not present
X'0C'	X'40'	X'0118'	LNKCTXS=NEW was given and the specified context is present
X'0C'	X'40'	X'011C'	A context specified with REFCTX@ is not present
X'0C'	X'01'	X'0120'	– An output area has read access only, is not assigned or is not aligned on a word boundary, – MPID is not assigned or is not aligned on a word boundary
X'0C'	X'01'	X'0121'	Invalid parameter list: One of the input parameters is wrong.
X'0C'	X'01'	X'0124'	An area that is to contain a name is not assigned. The name can be: – the object name (SYMBOL) – the name of the load unit (UNIT) – the version name (VERS) – the library name (LIBNAM) – the name of the REP file (REPFILE) – the link context name (LNKCTX) – a name in the list of reference context names (REFCTX)
X'0C'	X'01'	X'0125'	The number of reference context names is too big. No more than 200 names are permitted
X'0C'	X'01'	X'0127'	The mandatory operand SYMBLAD has not been specified.

X'cc'	X'bb'	X'aaaa'	Meaning
X'08'	X'01'	X'0129'	The length specified with USRMAPL is too small. Output of the DBL map to a user-defined data area is aborted without an end record. Loading is continued. This return code is output in the data area’s header.
X'0C'	X'01'	X'012A'	The length specified with USRMAPis too small.
X'08'	X'01'	X'012D'	The length specified with USRUNRL is too small. Output of the list of unresolved external references or of external references from earlier load operations (with UNRES=DELAY) which remained unresolved to a user-defined data area is aborted. Loading is continued. This return code is output in the data area’s header.
X'0C'	X'01'	X'012E'	The length specified with USRUNRL is too small.
X'0C'	X'01'	X'0134'	Illegal LNKCTXS operand
X'0C'	X'01'	X'0144'	Invalid link context name
X'0C'	X'01'	X'0146'	A reference context has the same name as the link context
X'0C'	X'01'	X'0148'	A link or reference context cannot be used
X'0C'	X'01'	X'014C'	An invalid symbol name consisting of blanks has been specified
X'0C'	X'01'	X'014D'	No symbol name has been specified
X'0C'	X'01'	X'0150'	Illegal entry in the SYMTYP operand. One of the following must be specified as the object type: CSECT (1) / ENTRY (2) /CSEN (3) / MODULE (4) / ANY (5). An object type other than those listed above was specified
X'0C'	X'20'	X'0158'	The maximum number of 16 user contexts has been reached. A new context cannot be generated
X'0C'	X'40'	X'016C'	Internal error in the AID debugging routine (system error)
X'0C'	X'40'	X'0184'	The identifier for MPID is invalid because – the memory pool (MP) was not set up or the user not connected to the MP – the MP is not in class 6 memory – the MP is already being used for the user' s shared code
X'0C'	X'01'	X'0188'	Invalid load address specified in LOAD@
X'0C'	X'40'	X'018C'	An attempt was made to use a context which has been corrupted by a previous error

X'cc'	X'bb'	X'aaaa'	Meaning
X'0C'	X'40'	X'0190'	An illegal residence mode (RMODE) has been defined for the load unit by DBL. This can happen when loading modules or COMMONs. Possible cause when loading a module: – The dummy RMODE defined for the module is 24 and the load unit is loaded into a memory pool (MP) above 16 Mb Possible causes when loading a COMMON: 1. The COMMON is to be loaded below 16 Mb and the load unit is loaded into a MP above 16 Mb. This is the case when: – the COMMON has RMODE 24 or – the load unit has PROGMOD 24 or – at least one module of the load unit is loaded below 16 Mb. 2. The COMMON was loaded in same context in a previous load unit above 16 Mb and is to be loaded below 16 Mb in the current load unit
X'0C'	X'01'	X'0194'	A class 6 memory area requested by DBL has been released by the user, or the attributes of memory pages have been changed by the user. An error occurred during the validity check of the released area.
X'0C'	X'20'	X'0196'	Common memory pool connection error. The task cannot be connected to the memory pool because (for example) part of the memory area for the memory pool has already been requested by the current task
X'0C'	X'20'	X'0198'	There is not enough space available for loading the objects specified by the user
X'0C'	X'20'	X'01A0'	A resource cannot be accessed because it is locked
X'0C'	X'40'	X'0200'	Error during DBL execution (system error)
X'0C'	X'40'	X'0204'	Inconsistencies in the DBL memory management tables (system error)
X'0C'	X'40'	X'0208'	Inconsistencies in the DBL data management tables (system error)
X'0C'	X'40'	X'020C'	Inconsistencies in the symbolic information tables (system error)
X'0C'	X'20'	X'0300'	Error in $REQM, $RELM, $CSTA, RDTFT (system error)
X'0C'	X'40'	X'0400'	Errors detected in object records during processing of a module (OM or LLM)
X'0C'	X'40'	X'0404'	Invalid type code in the ESD/ESV record
X'0C'	X'40'	X'0408'	Inconsistencies in the input module (e.g. in the order of ESD/ESV records)
X'0C'	X'40'	X'040C'	The specified module is not contained in the current load unit. Error in the RLD/LRLD record
X'0C'	X'40'	X'0410'	Error in the TXT record
X'0C'	X'40'	X'0414'	Error during processing of symbolic information
X'04'	X'40'	X'0418'	Error in REP record. Processing is continued. (Message BLS0230 was answered with YES in interactive mode.)
X'08'	X'40'	X'0418'	Error in REP record. Processing is continued (batch mode)

X'cc'	X'bb'	X'aaaa'	Meaning
X'0C'	X'40'	X'0418'	Error in REP record. Processing was aborted. (Message BLS0230 was answered with NO in interactive mode.)
X'04'	X'40'	X'041C'	The specified module is not in the current load unit. The REP record is ignored. Processing is continued. (Message BLS0234 answered YES)
X'0C'	X'40'	X'041C'	The specified module is not in the current load unit. The REP record cannot be processed. Processing was aborted. (Message BLS0234 was answered with NO.)
X'04'	X'40'	X'0420'	Error in the INCLUDE statement. Processing is continued. (Message BLS0230 was answered with YES.)
X'0C'	X'40'	X'0420'	Error in the INCLUDE statement. Processing was aborted
X'0C'	X'40'	X'0430'	The LLM cannot be loaded. Possible reasons: – It cannot be loaded with the current BLSSERV version. – Only one LLM with user-defined slices can be loaded in a context, and one already exists. – LLM can only be processed by BINDER and LLMAM . – LLM with CSECT AMODE=31 cannot be loaded with PROGMOD=24. – An LLM with user-defined slices cannot be loaded in a list name unit.
X'0C'	X'40'	X'0432'	An address resulting from a relocation was truncated. Processing is continued
X'0C'	X'40'	X'0433'	An address resulting from a instruction is outside the current segment. Processing is aborted
X'0C'	X'40'	X'0434'	This LLM is inconsistent with the “PRE-LOADING” option and preloading was requested. Processing is aborted
X'0C'	X'40'	X'0435'	The length transferred to DSSM for the preloadable section does not match the actual length of the preloadable LLM section
X'04'	X'01'	X'0600'	The module specified in an INCLUDE statement cannot be found. Processing is continued. (Message BLS0232 was answered with YES.)
X'0C'	X'01'	X'0600'	The module specified in an INCLUDE statement cannot be found. Processing was aborted. (Message BLS0232 was answered with NO.) or The module specified with SYMBOL@ or SYMBOL cannot be found in the specified libraries
X'04'	X'01'	X'0604'	A name conflict has occurred and has been accepted
X'0C'	X'01'	X'0604'	A name conflict has occurred. Processing was aborted
X'04'	X'01'	X'0608'	External references cannot be resolved. Processing is continued. (Message BLS0350 was answered with YES.)
X'08'	X'01'	X'0608'	Unresolved external references remain in the context and are processed later when possible. Processing is continued. (Return code issued only with `UNRES=DELAYWARN`.)

X'cc'	X'bb'	X'aaaa'	Meaning
X'0C'	X'01'	X'0608'	External references cannot be resolved. Processing was aborted. (Message BLS0350 was answered with NO.)
X'04'	X'01'	X'0609'	XDSECs-Rs cannot be resolved. Processing is continued
X'0C'	X'01'	X'0609'	XDSECs-Rs cannot be resolved. Processing was aborted
X'04'	X'40'	X'060C'	LOAD=NO was specified but the module is not loaded
X'0C'	X'40'	X'060D'	The symbol is already loaded in LNKCTX. It is not loaded again so as to avoid name conflicts
X'0C'	X'01'	X'0610'	The specified file is not a program library (PL) or an object module library (OML)
X'0C'	X'01'	X'0614'	The specified file is not a valid object module library (OML)
X'0C'	X'01'	X'0618'	The PLAM library routine is not available
X'0C'	X'01'	X'061C'	PAM read error in the library
X'0C'	X'01'	X'0620'	Error in the OML directory
X'0C'	X'40'	X'0624'	The load unit contains a CSECT with AMODE 31 and: – the hardware does not allow 31-bit addressing – PROGMOD=24 was specified or – the module with the specified symbol has already been loaded above 16 Mb and PROGMOD=24 was specified
X'04'	X'40'	X'0628'	The current addressing mode does not match the addressing mode that has to be used for the entry point (e.g. when the called program is executing with AMODE=31 and the entry point has AMODE=24). The addressing mode has to be switched over. However, DBL still passes the correct start address in the SYMBLAD field
X'0C'	X'40'	X'062C'	An attempt was made to load a module with RMODE=24 into a memory pool above 16 Mb
X'0C'	X'01'	X'0630'	An LLM containing slices for loading above and below 16 Mb cannot be loaded into a memory pool
X'0C'	X'01'	X'0634'	An LLM containing user-defined slices cannot be loaded into a memory pool
X'0C'	X'01'	X'0636'	The HSI codes of the ILE and server module are incompatible. The server module has not been loaded.
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported
X'00'	X'03'	X'FFFF'	The interface version is not supported

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Example

During the BIND1 program run, a **BIND** macro is issued to load a second CSECT, BIND2, as an overlay. BIND2 is stored as an LLM in library MACEXMP.LIB. Both CSECTs are to execute in 31-bit addressing mode. BIND1 is to be loaded below, and BIND2 above, the 16-Mb boundary. After the **BIND** macro is called BIND2 is to execute first.

Program BIND1

```
BIND1    START
         PRINT NOGEN
BIND1    AMODE 31                                                         (1)
BIND1    RMODE 24
         BALR  3,0
         USING *,3
         USING BINDSECT,6                                                 (2)
         ST    3,AREA11
         UNPK  AREAH,AREA1
         MVC   AREAA(8),AREAH
WROUT1   WROUT OUT,ERROR,PARMOD=31                                        (3)
BACK     LA    12,MVC
BIND     BIND  MF=E,PARAM=BINDPAR                                         (4)
         LA    6,BINDPAR
         CLC   XBINRET,=X'00000000'                                       (5)
         BE    MVC
         MVC   OUT+5(26),='BIND error!               '
         WROUT OUT,ERROR,PARMOD=31
         B     ERROR
MVC      MVC   OUT+5(26),='Back in BIND1             '
         WROUT OUT,ERROR,PARMOD=31                                        (6)
ERROR    TERM
OUT      DC    Y(OUTE-OUT)
         DS    CL3
         DC    C'BIND1: BASEREG.= '
AREAA    DS    CL8
OUTE     EQU   *
AREA     DS    0F
AREA1    DS    0CL5
AREA11   DS    CL4
AREA12   DC    C'0'
         DS    0F
AREAH    DS    CL9
BINDPAR  BIND  MF=L,SYMBOL=BIND2,SYMBLAD=BIND2@,BRANCH=YES,PROGMOD=ANY,*
               LIBLINK=PLAMLIB,MAP=SYSOUT
BIND2@   DS    A
BINDSECT BIND  MF=D,PREFIX=X                                              (7)
         END
```

Program BIND2

```
BIND2    CSECT -------------------------------------------------------- (8)
         PRINT NOGEN
BIND2    AMODE ANY ---------------------------------------------------- (9)
BIND2    RMODE ANY
         BALR  4,0
         USING *,4
         ST    4,AREA11
         UNPK  AREAH,AREA1
         MVC   AREAA(8),AREAH
         WROUT OUT,ERROR,PARMOD=31 ----------------------------------- (10)
         BR    12
ERROR    TERM
OUT      DC    Y(OUTE-OUT)
         DS    CL3
         DC    C'BIND2: BASEREG.= '
AREAA    DS    CL8
OUTE     EQU   *
AREA     DS    0F
AREA1    DS    0CL5
AREA11   DS    CL4
AREA12   DC    C'0'
AREAH    DS    CL9
         END
```

(1) The AMODE=31 attribute is defined for program segment BIND1. The RMODE=24 attribute means that BIND1 will always be loaded below the 16Mb boundary.

(2) Register 6 is assigned by the assembler as the base address register for addressing the DSECT for the operand list of the **BIND** macro, which is generated at the symbolic address BINDSECT as a result of a **BIND** macro specifying MF=D.

(3) The contents of the base register for BIND1 are output to indicate the addressing mode and the load address.

(4) The **BIND** macro is called in its E form at the symbolic address BIND. At this point in the program, therefore, only the instruction part is generated. The associated operand list is created at the symbolic address BINDPAR by means of a **BIND** macro specifying MF=L. As a result of the operand values specified in the list, the **BIND** macro causes the following to happen at program runtime:
- the CSECT BIND2 (SYMBOL=BIND2,SYMTYP=CSECT) is reloaded from the library assigned with the link name PLAMLIB (LIBLINK=PLAMLIB),
- the start address of BIND2 is stored in field BIND2@ (SYMBLAD=BIND2@),
- the 31-bit addressing mode is set for BIND2 (PROGMOD=ANY),
- a DBL map is output to SYSOUT (MAP=SYSOUT), and
- the program run is continued in BIND2 after BIND2 has been loaded (BRANCH=YES).

(5) Following execution of the **BIND** macro, a check is made to verify that the XBINRET field of the standard header contains the return code X'00000000', which indicates error-free execution of the macro. The name XBINRET originates from the DSECT that was generated under the symbolic address BINDSECT as a result of a **BIND** macro specifying MF=D and PREFIX=X (see (7)). This DSECT describes the layout of the operand list of the **BIND** macro. The symbolic names of the DSECT can be used for addressing within the operand list once the assigned base address register (in this case, register 6) has been loaded with the start address of the operand list (in this case, BINDPAR).

(6) A message is written to SYSOUT to indicate that the program run is continued in BIND1 following execution of BIND2.

(7) The **BIND** macro specifying MF=D generates a DSECT which describes the layout of the operand list of the **BIND** macro. The PREFIX=X operand causes the letter X to be prefixed to all symbolic names in this DSECT (field names and equates).

(8) The CSECT statement defines the program segment BIND2.

(9) AMODE=ANY indicates to the operating system that BIND2 can execute in 24-bit or 31-bit addressing mode.

(10) The contents of the base register for BIND2 are output to indicate the addressing mode and the load address.

Runtime log

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,bind1), - ——————————— (11)
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,bind1))
%  ASS6011 ASSEMBLY TIME: 557 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 84 MSEC
//compile source=*library-element(macexmp.lib,bind2), - ——————————— (12)
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,bind2))
%  ASS6011 ASSEMBLY TIME: 270 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 82 MSEC
//end
%  ASS6012 END OF ASSEMBH
```

```
/add-file-link link-name=plamlib,file-name=macexmp.lib ------------- (13)
/start-executable-program library=macexmp.lib, - ------------------ (14)
/      element-or-symbol=bind1,prog-mode=*any
%  BLS0523 ELEMENT 'BIND1', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'BIND1', VERSION ' ' OF '<date> <time>' LOADED
BIND1: BASEREG.= 80000002 ----------------------------------------- (15)
------------------------------------------------------------------- (16)
############################### A D B L   M A P ############################
#                                                                          #
# LOAD UNIT: BIND2. . . . . . . . . . . . . . LOAD INFO  =DEF            #
# VERSION  : ~                                LOAD TIME  =<date> <time>  #
# CONTEXT  : LOCAL#DEFAULT. . . . . . . . . . TEST OPTION=NONE           #
#                                                                          #
#  LLM     : BIND2. . . . . . . . . . . . . . BY_ATTR  / EXPLICIT  PLAMLIB #
#   OM     : BIND2. . . . . . . . . . . . . . STANDARD                   #
#    CSECT : BIND2. . . . . . . . . . . . . . @= 1000000 L=      7E      #
#                                                                          #
# LOAD UNIT STARTING POINT. . . . . . . . . . @= 1000000 AMODE=31 HSI=/7500 #
#                                                                          #
######################## E N D   O F   A D B L   M A P ####################
BIND2: BASEREG.= 81000002 ----------------------------------------- (17)
Back in BIND1 ----------------------------------------------------- (18)
```

(11) Program BIND1 is assembled.

(12) Program BIND2 is assembled.

(13) The file link name used in the BIND call (4) is assigned.

(14) DBL is invoked to link, load and start the program.

(15) The contents of the base register for BIND1 are output. 31-bit addressing is set (bit 2^{31} = 1); the load address is below the 16-Mb boundary.

(16) DBL has loaded the CSECT BIND2. A DBL log is written to SYSOUT.

(17) The contents of the base register for BIND2 are output. 31-bit addressing is set (bit 2^{31} = 1); the load address is above the 16-Mb boundary.

(18) Following the return from BIND2, the program run is continued in BIND1.

BKPT – Set breakpoint; interrupt current program

General

Application areas:	Debugging aids; see page 162 Interrupting the program run; see page 72 Communication; see page 163
Macro type:	O-Type O; see page 28

Macro description

The **BKPT** macro serves to interrupt program execution in order to enter commands.
In interactive mode, system commands or IDA commands can be input at the terminal. The program is continued with the RESUME-PROGRAM, %RESUME or %TRACE command.
In the case of batch jobs and interactive procedures, commands are processed until the next RESUME-PROGRAM, %RESUME or %TRACE command is encountered.

Macro format and description of operands

BKPT

Example

```
BKPT     START
         PRINT NOGEN
         BALR  3,0
         USING *,3
         WROUT MESS1,ERROR  ——————————————————————————————————————————  (1)
         BKPT  ———————————————————————————————————————————————————————  (2)
         WROUT MESS1,ERROR  ——————————————————————————————————————————  (3)
ERROR    TERM
***  Definitions   ****
MESS1    DC    Y(END-MESS1)
         DS    L2
         DC    X'01'
         DC    C'Here is BKPT'
END      EQU   *
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,bkpt), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,bkpt))
%  ASS6011 ASSEMBLY TIME: 314 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 80 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=bkpt
%  BLS0523 ELEMENT 'BKPT', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'BKPT', VERSION ' ' OF '<date> <time>' LOADED
Here is BKPT ──────────────────────────────────────────────────────────  (1)
%  IDA0199 PROGRAM BREAK AT ADDRESS X'000014', AMODE=24 ───────────────  (2)
/show-user-status inf=*prog
NAME      TSN TYPE        SIZE CURR-CMD
MACTEST  2QSE 3 DIALOG1      1 SHOW-USER-STATUS
              PROG::2OSG:$QM212.MACEXMP.LIB(BKPT,@,L)
/resume-program
Here is BKPT ──────────────────────────────────────────────────────────  (3)
```

(1) A message is written to SYSOUT.

(2) The **BKPT** macro is encountered: program execution is interrupted. The message `IDA0199` is output.

Commands can now be input at the user terminal. In this example, the SHOW-USER-STATUS INF=*PROG command is input. With the RESUME-PROGRAM command the interrupted program is continued from the line after **BKPT**.

(3) A message is written to SYSOUT.

CALL – Load segments

General

Application area:	Linking and loading; see page 47
Macro type:	O-Type O; see page 28

Macro description

The **CALL** macro enables the user to load a segment that is not yet in memory by specifying a symbol. Subsequent segments within the same path of the overlay structure are loaded automatically.

After loading is completed program execution continues at the specified symbol in the segment just loaded.

Macro format and description of operands

CALL
symbol

symbol
Symbolic address (entry point) within the segment to be loaded.
A 4-byte V-type constant is generated for this symbol. This operand allows the user to implicitly specify the segment to be loaded and, simultaneously, to specify the address at which program execution continues after macro execution.

Functional description

The **CALL** macro (and also the **SEGLD** macro) implements the automatic loading of segments (nonautomatic loading is performed by the **LPOV** macro).

An instruction in the **CALL** macro causes the Assembler to generate a V-type constant from the specified symbolic address. This constant identifies both the segment to be loaded and the address at which program execution continues after loading is completed. Based on this V-type constant in the **CALL** macro and on the CONTROL=YES operand in the PROGRAM control statement (see the “Utility Routines” manual [27]), the linkage editor generates an overlay control module, required for automatic loading, which satisfies the V-type constant with an address. The **CALL** macro includes a branch to the V-type constant. When **CALL** is to be executed, control is passed to the overlay control module, which performs a check to establish whether the required segment is already in memory. If the segment has already been loaded, program execution will continue at the instruction identified by the operand “symbol”. If the segment specified has not yet been loaded, it will be brought into memory together with all segments that are in the same path. After the loading operation the program will continue - in the same way as if no loading had occurred - at the specified address in the segment to be loaded.

Notes

- The overlay structure of a program in which automatic loading of segments is performed should be designed in such a way as to allow for additional memory as required for the overlay control module, for ENTAB and for SEGTAB (see the “Utility Routines” manual [27]).
- The **CALL** macro must be contained in a program area covered by a USING statement. As register 15 is used by the **CALL** macro for segment loading (it contains the entry address), it must not be used as the base register for the program section that contains the **CALL** macro.

CDUMP2 – Generate user, system or area dump

The three types of dump supported by **CDUMP2** differ in the scope of the information offered and in the name and the ID for the dump file output. The following sections outline the most important characteristics of area, user, and system dumps:

Area dump

Scope of the area dump:

- The areas of the user address space from class 6 memory and class 5 memory which are specified in the **CDUMP2** call, provided the output is not prohibited by the system parameter DUMPCL5P.
- Specially protected pages ("secret pages", **CSTAT** PROTECT=YES macro call) are output only in the scope which is specified by the system parameter DUMPSEPA.
- Other system areas: module AIDSYSD, the area with COMAREA, the TCB and TU-PCB tables (see user dump); in addition - if requested by the MODE=*EXP operand - all other system areas of the user dump.
- Data spaces (see section "Extended addressing with data spaces" on page 61) to which the **CDUMP2** caller is connected. The caller uses the data area to determine from which areas data spaces are to be saved.
- The output can be controlled with /MODIFY-TEST-OPTIONS USERDUMP-OPTIONS=*PARAMETERS(...), see the "commands" manual [19].

ID for the area dump file:

- User ID of the task which has called **CDUMP2**, if the user is authorized to read all the data output in the dump.
- SYSUSER system ID if the dump contains read-protected data which the user is not authorized to access (e.g. programs protected by a read password which the user has not entered in the password table of the task).

Name of the area dump file:

- `$userid.SYS.ADUMP[.jobname].tsn.i` if the file is entered under the ID of the **CDUMP2** caller.
- `$SYSUSER.SYS.ADUMP[.jobname].tsn.i.userid` if the file is entered under the SYSUSER system ID.

  Where:

`userid`	User ID of the **CDUMP2** caller
`jobname`	JOB-NAME of the job which has called **CDUMP2** (from the SET-LOGON-PARAMETERS command)
`tsn`	TSN of the job which has called **CDUMP2**
`i`	Consecutive number of the area/user dump within the task

User dump

Scope of the user dump:

- Total class 6 and class 5 memory, provided this is not prohibited by the system parameter DUMPCL5P.
- Specially protected pages (“secret pages”, **CSTAT** PROTECT=YES macro call) are output only in the scope which is specified by the system parameter DUMPSEPA.
- Other system areas: AIDSYSD module, Trace Dump List; task-specific system tables such as TCB (Task Control Block), TFT (Task File Table), TU-PCB (Process Control Block), TU-AUDIT-TABLE (AUDIT table, see **AUDIT** macro).
- The system trace table is buffered when **CDUMP2** is called and is included in the dump when the dump is created (at the point where it occurs in the system).
- If the error task program counter (PC) indicates the system address space, any pages referenced via general purpose registers and PCs (plus 5 pages preceding and 5 pages following) are also output. If at least one page is present which is privileged but not “common readable”, the user dump is output to the ID SYSUSER.
- Data spaces (see section “Extended addressing with data spaces” on page 61) to which the **CDUMP2** caller is connected.
  The caller uses the DS macro operand to determine whether only specific data spaces, all data spaces or no data spaces are to be saved. If the DS operand is not specified, the value (YES or NO) set in the MODIFY-TEST-OPTIONS command, operand DATA-SPACES=*YES/*NO determines the output of data spaces.
- The output can be controlled with /MODIFY-TEST-OPTIONS USERDUMP-OPTIONS=*PARAMETERS(...), see the “commands” manual [19].

ID for user dump file:

- User ID of the task which has called **CDUMP2** if the user is authorized to read all the data output in the dump and if the dump does not contain referenced pages located in privileged address space and which are not “common readable”.
- $SYSUSER system ID if the dump contains read-protected data which the user is not authorized to access (e.g. programs protected by a read password which the user has not entered in the password table of the task).

Name of the user dump file

- `$userid.DUMP[.jobname].tsn.i` if the file is not created under the user ID of the **CDUMP2** caller.
- `$SYSUSER.DUMP[.jobname].tsn.i.userid` if the file is not entered under the $SYSUSER user ID.

  For function, see area dump.

System dump

Users wishing to request a system dump must have set their read privileges to a value m≥3 with /MODIFY-TEST-OPTIONS PRIVILEGE=*PARAMETERS(READ=m,WRITE=1). They are only authorized to do so if this privilege is granted to them in the user catalog.

A user or area dump can be converted to a system dump if the test privilege has been granted (see above)

- using /MODIFY-TEST-OPTIONS, operand DUMP=*SYSTEM
  The message `IDA0N45` is suppressed.
- by answering the message `IDAN045` with `Y,SYSTEM`

The operator can control whether the system dump is output to disk, or to tape or magnetic tape cartridge.
If a system dump is abnormally terminated, the message `IDA0N99` is output depending on the value of the DUMPCTRL system parameter.

Scope of the system dump:

- Total class 6 memory - depending on the value of the system parameter DUMPSREF - except the pages which were declared as “secret pages”
- Total class 5, class 3 and class 1 memory except the pages which were declared as “secret pages”.
- All data areas from class 4 memory and those pages from class 4 and class 2 memory, covering the 5 pages before and after an address, to which the program counters (PC) from the PCBs and the trace table, and the general purpose registers of the PCBs and the bourse registers of the TCB refer. Exceptions to this are “secret pages”.
- Specially protected pages (“secret pages”, **CSTAT** PROTECT=YES macro call) are output only in the scope which is specified by the system parameter DUMPSEPA.
- With SNAP, class 1, class 3, and resident class 4 memory is buffered before the dump is actually generated and is then included in the dump.
- The system trace table is buffered when **CDUMP2** is called and is included in the dump when the dump is created (at the point where it occurs in the system).
- Additional system areas: the modules AIDSYSD, EOLDTAB, DMCHD, NSISINF, CLASS2OP; areas with TU-AUDIT, Trace Dump List and the REPLOG and SERSLOG system files; system tables as in user dump.
- Data spaces (see section “Extended addressing with data spaces” on page 61) that the operating system has defined in the dump creator CDUMP via a privileged interface ($DMPDEF(I)).

ID for system dump file: $SYSDUMP

Name of the system dump file:

```
                  ⎧ABSOLU ⎫
:catid:$SYSDUMP.⎨        ⎬.pc.ec.tsn.date.time
                  ⎩module ⎭
```

Where:

`modul`	Name of the module from which **CDUMP2** is called, max. 8 characters
`ABSOLU`	is used if there is no name for the module
`pc`	Address in the program counter (relative to the start of the module)
`ec`	Event code (hexadecimal)
`tsn`	TSN of the task which has called **CDUMP2**.
`date`	date in the form Dyymmdd (D draws attention to the begin of the date)
`time`	Time in the form hhmmss

Dump output

- The user can request a dump if /MODIFY-TEST-OPTIONS USERDUMP-OPTIONS=*PARAMETERS(DUMP=*YES) was previously specified or if DUMP=*STD is the default setting. With DUMP=*YES, the dump will be automatically created, otherwise the following query is issued in response to a dump request:

  ```
  IDA0N45 DUMP DESIRED? REPLY (Y=USER-/AREADUMP TO DISK;
                               Y,<VOLUMETYPE>=USER-/AREADUMP TO TAPE;
                               Y,SYSTEM=SYSTEMDUMP TO DISK;
                               N=NO)
  ```

  The user is offered the choice between a user or area dump (response: Y) or a system dump (response: Y,SYSTEM). Only a user who has adequate test privileges can request a system dump. If `<VOLUMETYPE>` is entered, the dump is output to tape.

- If /MODIFY-TEST-OPTIONS USERDUMP-OPTIONS=*PARAMETERS(DUMP=*STD) is set, no memory dump is issued in procedures or in batch mode. The following message appears:

  ```
  IDA0N48 TASK/SYSTEM SETTINGS PROHIBIT DUMP
  ```

- If /MODIFY-TEST-OPTIONS USERDUMP-OPTIONS=*PARAMETERS(DUMP=*NO) is set, a dump request is rejected. The following message appears:

  ```
  IDA0N47 DUMP PROHIBITED BY /MODIFY-TEST-OPTIONS COMMAND
  ```

- If the user has requested a system dump, the following message is displayed at the console:

  ```
  IDA0N52 SYSTEM DUMP DESIRED? REPLY (EOT=OUTPUT TO DISK;
                                      <VOLUMETYP>=OUTPUT TO TAPE;
                                      N=NO)
  ```

The console message `IDA0N52` can be suppressed with the system parameters DUMPCTRL and DUMPSD#. In general the message is suppressed if DUMPCTRL is set to “unattended operation” or “error in system task”. In addition, DUMPSD# can be used to specify the number of `IDA0N52` messages that are not to appear at the console in each session.

If `IDAN052` is suppressed, the dump is created automatically. This enables greater consistency in the data in the class 3 and class 4 memories.

- If the memory space available for output of a user or area dump is insufficient, the dump is interrupted with the following message:
  `IDA0N57 INSUFFICIENT DISK SPACE. NO DUMP OUTPUT`
- If the **CDUMP2** operands have not been correctly entered, the dump request is rejected with the message
  `IDA0N46 CDUMP OPERAND LIST INCORRECT OR NOT AVAILABLE.`
  If an error occurs in the system during output of the dump, the following messages – dependent on the error – are output:
  - `IDA0N63 DMS xxxx ERROR OCCURRED. DUMP PROCESSING CONTINUED`
    Dump processing is continued
  - `IDA0N61 DUMP PROCESSING ABORTED DUE TO aaa ERROR AT PC=bbb. RC=ccc`
    With a user or system dump, this message is also output to the console and a SERSLOG entry is written simultaneously. With a system dump, the SERSLOG file is included in the dump. Dump processing is aborted.
- Dump output is time-monitored. If the dump cannot be output within 36 minutes, output is aborted.
- Copying of files dumped to tape
  If files have been dumped to magnetic tape cartridge, they can be copied to disk by means of the COPY-FILE command. The utility routine PERCON cannot be used in this case.

Example

```
/IMPORT-FILE SUPPORT=*TAPE(VOLUME=<volume>,DEVICE=<device>,-
       FILE-NAME=<original filename>)

/ADD-FILE-LINK LINK=DMCOPY11,FILE-NAME=<original filename>,-
       ACCESS-METHOD=*UPAM,BUF-LEN=*STD(2)

/ADD-FILE-LINK LINK=DMCOPY22,FILE-NAME=<output filename>,-
       ACCESS-METHOD=*UPAM,BUF-LEN=*STD(2)

/COPY-FILE FROM-FILE=<original filename>,TO-FILE=<output filename>
```

See also COPY-FILE command in the “commands” manual [19].

General

Application area: Debugging aids; see page 162
Macro type: Type S, MF format **1**: C/D/L/M/E form; see page 29

CDUMP2 always generates a 31-bit interface. To generate a 24-bit interface, the old macro CDUMP (Appendix) must be used.

Macro description

The macro **CDUMP2** generates a dump (in a separate dump task) for the task which has called **CDUMP2**. By specifying the SCOPE operand, the user can determine whether an area dump, a user dump or a system dump is to be output.

The dump is written to the disk unedited as a PAM file (it may be written to magnetic tape or to magnetic tape cartridge). The dump cannot be distributed over several tapes. For analysis and printout the dump can be edited with the software products AID and DAMP (see the “AID” manual [3] and “Diagnostics Handbook” [9]).

Macro formats and description of operands

CDUMP2
MF=C / D / L / M / E
,SCOPE=*USER / *SYSTEM / *AREA
[,XPAND=PARAM / DSCB / AREA]
,PC=*STD / <var: pointer> / (<reg: pointer>)
,EC=*STD / <var: pointer> / (<reg: pointer>)
,CODE=<var: pointer> / (<reg: pointer>)
,INSERT=<var: pointer> / (<reg: pointer>)
,TITL=*STD / <var: pointer> / (<reg: pointer>)
,SNAP=*STD / *YES / *NO
,ELSN=*NONE / <var: pointer> / (<reg: pointer>)
,DIAG=*NO / *YES
,DIV=*STD / *YES / *NO
,DS=*STD / *YES / *NO / <var: pointer> / (<reg: pointer>)
,MMAP=*STD / *YES / *NO
,NUM=<integer 1..2048>
,MODE=*STD / *EXP
,DSCTRL=<var: pointer> / (<reg: pointer>)

The operands are described in alphabetical order below.

CODE=
designates a character sequence for identification of the dump;. length = 7 bytes. The character sequence is output in the `IDA0N51` message. This operand is only permitted in conjunction with SCOPE=*USER or SCOPE=*SYSTEM.

<var: pointer>
Name of the field with the address of the character sequence; may only be specified in conjunction with MF=M.

(<reg: pointer>)
Register with the address of the character sequence; may only be specified in conjunction with MF=M.

DIAG=
Determines whether the message `IDA0N50` indicating the address of the CDUMP2-SVC is sent to the operator. This operand may be specified only in conjunction with SCOPE=*SYSTEM.

***NO**
The message `IDA0N50` is not output.

***YES**
The message `IDA0N50` is output.

DIV=
Specifies whether DIV windows are to be included in the user dump. This operand may be specified only in conjunction with SCOPE=*USER, otherwise the value DIV=*YES is used. See also the section “Extended addressing with data spaces” on page 61.

***STD**
The value set in /MODIFY-TEST-OPTIONS, operand DATA-IN-VIRTUAL determines whether DIV windows are to be included in the user dump (*YES) or not (*NO).

***YES**
All DIV windows are to be included in the user dump.

***NO**
No DIV windows are to be included in the user dump.

DS=
Determines which memory areas of data spaces (DS) are to be included in the user dump. This operand may only be specified in conjunction with SCOPE=*USER.
The areas of data spaces included in the area dump are the ones that the user has specified in the DSCTRL operand.
In system dumps, only the areas of data spaces that the operating system has defined in in the dump creator CDUMP via a privileged interface ($DMPDEF(I)) are included.
See also the section “Extended addressing with data spaces” on page 61.

STD
The value set in /MODIFY-TEST-OPTIONS, operand DATA-SPACES determines whether data spaces are to be included in the user dump (*YES) or not (*NO).

***YES**
All data spaces (up to 100 data spaces used by the caller) are to be included in the user dump.

***NO**
No data spaces are to be included in the user dump.

<var: pointer>
Name of the field with the address of the list of data spaces; may only be specified in conjunction with MF=M.

(<reg: pointer>)
Register with the address of the list of data spaces; may only be specified in conjunction with MF=M.
This list contains the SPIDs (8 bytes per entry). The list must end with a zero entry (D(0)).

DSCTRL=
Points to a data space control block (DSCB). A data space and its associated areas can be defined in the DSCB. DSCTRL must point to the first DSCB. This operand may be specified only in conjunction with SCOPE=*AREA.

Data structure of a DSCB:

```
CDDDSCB    DSECT ,        DATA SPACE CTRL BLOCK
CDDDS@     DS    A                   Points to the next DSCB
CDDSPID    DS    XL8                 SPID of the DS
CDDNUM     DS    H                   Number of areas in the DS
CDDSTRT    DS    A                   Start of the first area
CDDEND     DS    A                   End of the first area
```

The maximum number of areas in the DSCB is 2048. By chaining DSCBs, users can specify areas of more than one data space of their task. By default, no areas of a data space are specified. See also the section “Extended addressing with data spaces” on page 61.

<var: pointer>
Name of the field with the address of the DSCB; may only be specified in conjunction with MF=M.

(<reg: pointer>)
Register with the address of the DSCB; may only be specified in conjunction with MF=M.

The following specification obtains the DSECT of a DSCB: CDUMP2 MF=D,XPAND=DSCB
The following specification obtains the DSECT of an area: CDUMP2 MF=D,XPAND=AREA

EC=

defines the location from where the event code (interrupt weight) is to be fetched. The event code is output in the `IDA0N51` message. This specification is permissible only in conjunction with SCOPE=*USER or SCOPE=*SYSTEM; if SCOPE=*AREA the event code is always fetched from the calling stack.

<u>*STD</u>

The event code is to be fetched from the caller stack

<var: pointer>

Name of the field with the address of the event code; may only be specified in conjunction with MF=M.

<reg: pointer>

Register with the address of the event code; may only be specified in conjunction with MF=M.

ELSN=

Specifies the address of a field containing the number of an Error Log Sequence Block to which the caller has written specific data in the error log file. The number is output in the `IDA0N51` message..

Field length = 4 bytes, which must be aligned on a word boundary. The number must be entered as a binary digit. This operand is only permitted in conjunction with SCOPE=*SYSTEM.

***NONE**

An ELSN is not output.

<var: pointer>

Name of the field with the address of the ELSN; may only be specified in conjunction with MF=M.

(<reg: pointer>)

Register with the address of the ELSN; may only be specified in conjunction with MF=M.

INSERT=

defines a text to be output with the message `IDA0N51`. This text could contain more detailed information on the cause of the dump.

The user must place this text, which may be up to 60 characters long, in a data area with the following format:

Byte 1: Length (hexadecimal) of the text to be output (in bytes). If byte 1 has the value 0, no INSERT text is output.

Byte 2 through n (n≤61): Text to be output.

This operand is only permitted in conjunction with SCOPE=*USER or SCOPE=*SYSTEM.

<var: pointer>
Name of the field with the address of the text; may only be specified in conjunction with MF=M.

(<reg: pointer>)
Register with the address of the text; may only be specified in conjunction with MF=M.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under "Macro type" and are included in the macro format.
The following operands must be specified for area dumps: The SCOPE operand if MF=C/D/E, and additionally the NUM operand if MF=C/D.
For system dumps, the SCOPE operand is mandatory if MF=C/D/E.

MMAP=
Defines whether memory map pages (data areas for POSIX pages) are to be stored in the dump. This operand is only evaluated if SCOPE=*USER is also specified. If SCOPE=*SYSTEM is specified, MMAP=*YES is assumed.

<u>*STD</u>
Default value: the value specified in /MODIFY-TEST-OPTIONS, operand MMAP determines whether memory map pages are present in the user dump.

***YES**
All the memory map pages associated with the task are to be included in the user dump.

***NO**
No memory map pages are to be present in the user dump.

MODE=
Defines the scope of the diagnostic data to be output for an area dump. The operand may be specified only in conjunction with SCOPE=*AREA.

<u>*STD</u>
Ensures that only the AIDSYSD module and the areas with COMAREA, TU-PCB and TCB are dumped in addition to the specified areas in class 6 and class 5 memory (see the section "Area dump" on page 278).

***EXP**
Initiates dumping of the area with COMAREA and all other system areas as in user dump in addition to the areas specified in class 6 and class 5 memory (see the section "User dump" on page 279).

NUM=
Specifies the number of areas to be dumped. NUM is permissible only in conjunction with MF=L/C/D and only in conjunction with SCOPE=*AREA The start and end addresses of the areas to be dumped must be entered (dynamically) in the generated data area. The DSECT of the area is generated with `CDUMP2 MF=D,XPAND=AREA`. If MF=C/D, NUM is mandatory.

<integer 1..2048>
Number of areas to be dumped as a direct specification; number range: 1..2048.

PC=
designates a register or field which contains the program counter to be logged.

***STD**
The program counter is to be fetched from the calling stack.

<var: pointer>
Name of the field with the address of the program counter; may only be specified in conjunction with MF=M.

(<reg: pointer>)
Register with the address of the program counter; may only be specified in conjunction with MF=M.

SCOPE=
specifies whether a user, system or area dump is to be output.

***USER**
A user dump is output

***SYSTEM**
A system dump is output. This specification may only be entered by users with a read test privilege of ≥ 3 or for TPR programs

***AREA**
An area dump is output. The addresses for the areas must be specified immediately after the parameter list. Depending on the number of areas the addresses must be specified in the following sequence: start address, end address, start address, end address, ... start address, end address.

SNAP=
determines the use of the SNAP to generate a highly consistent system dump. This operand is only permitted in conjunction with SCOPE=*SYSTEM

***STD**
Means that an associated snap dump, if one exists, is used to create the system dump (if the system dump occurred because of a TPR program error, the system will normally have initiated a SNAP dump for it).

***YES**
means that CDUMP2 generates a snap dump (unless an associated one already exists) and uses this to create the system dump.

***NO**
means that CDUMP2 creates the system dump without using an associated snap dump (even though one may exist).

TITL=
defines an additional title line for the memory dump. Length = 132 characters.

<u>*STD</u>
CDUMP2 generates a standard title line.

<var: pointer>
Name of the field with the address of the title line; may only be specified in conjunction with MF=M.

(<reg: pointer>)
Register with the address of the title line; may only be specified in conjunction with MF=M.

XPAND=
specifies whether a DSECT is to be generated for the CDUMP2 parameter list, the DSCB, or an area. May only be specified in conjunction with MF=D.

PARAM
A DSECT is generated for the CDUMP2 parameter list.

DSCB
A DSECT is generated for the data space control block.

AREA
A DSECT is generated for an area.

The following specification obtains the DSECT of a DSCB: CDUMP2 MF=D,XPAND=DSCB
The following specification obtains the DSECT of an area: CDUMP2 MF=D,XPAND=AREA

Return information and error flags

The return codes are compatible with those of the old CDUMP macro.

Standard header:

						a	a

A return code relating to the execution of the CDUMP2 macro is transferred in the standard header (aa=main code).

X'aa'	**Meaning**
X'00'	Dump completed without errors
X'04'	Dump completed using default values
X'08'	Dump suppressed due to /OPTION DUMP operand
X'0C'	Dump suppressed due to system standards
X'10'	Dump suppressed due to severe CDUMP2 operand error
X'14'	Dump suppressed due to insufficient test privilege
X'18'	Dump suppressed due to error in DMS routines
X'1C'	Dump suppressed due to system error
X'20'	Dump suppressed due to previous interruption of CDUMP2
X'28'	Dump suppressed due to SHUTDOWN processing
X'2C'	Dump suppressed by calling task
X'30'	All area specifications are invalid. No dump is output
X'34'	Incorrect specification for NUM=... . No dump is output
X'38'	Some dump areas are not located in the caller' s address space or the specifications are inconsistent. At least one memory area has, however, been output
X'3C'	Dump suppressed since “DMS READY” has not yet been reached

Example

The following program creates a memory pool in class 6 memory and generates a user dump and an area dump.

```
CDUMP2   START
         PRINT NOGEN
         BALR  3,0
         USING *,3
         ENAMP MPNAME=MEMP,SCOPE=GLOBAL,MPIDRET=PID,BSIZE=48 ──────────  (1)
REQMP    REQMP MPID=PID,BSIZE=5 ───────────────────────────────────────  (2)
         LA    4,4095(1)
         LA    4,1(4)
         ST    4,AR1ANF
         MVC   0(TEXT1LEN,4),TEXT1 ────────────────────────────────────  (3)
         LA    4,4095(4)
         ST    4,AR1END
         LA    4,1(4)
         ST    4,AR2ANF
         MVC   0(TEXT2LEN,4),TEXT2 ────────────────────────────────────  (4)
         LA    4,4095(4)
         ST    4,AR2END
CDUMPUS  CDUMP2 MF=E,PARAM=PARAM ──────────────────────────────────────  (5)
CDUMPAR  CDUMP2 MF=E,PARAM=AREA ───────────────────────────────────────  (6)
TERM     TERM
****  Definitions  ****
         DS    0F
PID      DS    F
TEXT1    DC    C'AREADUMP: '
         DC    C'PAGE 2 OF MEMORY POOL '
         DC    100C'A'
         DC    100C'B'
TEXT1LEN EQU   *-TEXT1
TEXT2    DC    C'AREADUMP: '
         DC    C'PAGE 3 OF MEMORY POOL '
         DC    100C'C'
         DC    100C'D'
TEXT2LEN EQU   *-TEXT2
PARAM    CDUMP2 MF=L,SCOPE=*USER ──────────────────────────────────────  (7)
AREA     CDUMP2 MF=L,SCOPE=*AREA,NUM=2,MODE=*EXP ──────────────────────  (8)
AR1ANF   DS    A ──────────────────────────────────────────────────────  (9)
AR1END   DS    A
AR2ANF   DS    A
AR2END   DS    A
         END
```

(1) A memory pool consisting of 48 main memory pages is set up above the 16 Mb boundary. The start address of the memory pool is stored in Register R1.

(2) Five main memory pages are required from the start address of the main memory pool.

(3) The text `AREADUMP: PAGE 2 OF MEMORY POOL` and 100 times A and B are entered in the memory pool.

(4) The text `AREADUMP: PAGE 3 OF MEMORY POOL` and 100 times C and D are entered in the memory pool.

(5) The **CDUMP2** macro is called in its E form at the symbolic address of the program, CDUMPUS. The associated data area is generated at the symbolic address PARAM by a **CDUMP2** call specifying MF=L (see (7)).

The operands specified there cause the macro to output a user user dump when it is executed (SCOPE=*USER).

(6) The **CDUMP2** macro is called again in its E form at the symbolic address of the program, CDUMPAR. In this case the associated data area is generated at the symbolic address AREA by a **CDUMP2** call specifying MF=L (see (8)).
The operands specified there cause the macro to output an area dump for two areas (NUM=2) when executed in its extended form.

(7) Data area for the **CDUMP2** call in (5).

(8) Data area for the **CDUMP2** call in (6).

(9) The addresses for the areas must be specified immediately after the parameter list. Depending on the number of areas the addresses must be specified in the following sequence: start address, end address, start address, end address, ... start address, end address.

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,cdump2), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,cdump2))
%  ASS6011 ASSEMBLY TIME: 492 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 82 MSEC
//end
%  ASS6012 END OF ASSEMBH
```

```
/start-executable-program library=macexmp.lib,element-or-symbol=cdump2
%  BLS0523 ELEMENT 'CDUMP2', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'CDUMP2', VERSION ' ' OF '<date> <time>' LOADED
%  IDA0N51 PROGRAM INTERRUPT AT LOCATION '0000006A (CDUMP2), (CDUMP)'
%  IDA0N45 DUMP DESIRED? REPLY (Y=USER-/AREADUMP TO DISK;
   Y,<VOLUMETYPE>= USER-/AREADUMP TO TAPE; Y,SYSTEM=SYSTEMDUMP;
   N=NO)? y ──────────────────────────────────────────────────────── (10)
%  IDA0N53 DUMP BEING PROCESSED. PLEASE HOLD ON
%  IDA0N54 'USERDUMP' WRITTEN TO FILE '$QM212.DUMP.V.2RCU.00001' ────── (11)
%  IDA0N55 TITLE: 'TSN-2RCU  UID-QM212  AC#-89002  USERDUMP  PC-0000006A
   EC-50  VERS-150  DUMP-TIME  13:47:16  12-01-20'
%  IDA0N51 PROGRAM INTERRUPT AT LOCATION '00000070 (CDUMP2), (ADUMP)'
%  IDA0N45 DUMP DESIRED? REPLY (Y=USER-/AREADUMP TO DISK;
   Y,<VOLUMETYPE>=USER-/AREADUMP TO TAPE; Y,SYSTEM=SYSTEMDUMP;
   N=NO)? y ──────────────────────────────────────────────────────── (12)
%  IDA0N53 DUMP BEING PROCESSED. PLEASE HOLD ON
%  IDA0N54 'AREADUMP' WRITTEN TO FILE '$QM212.SYS.ADUMP.V.2RCU.00002' ─ (13)
%  IDA0N55 TITLE: 'TSN-2RCU  UID-QM212  AC#-89002  AREADUMP  PC-00000070
   EC-50  VERS-150  DUMP-TIME  13:47:22  12-01-20'
```

(10) A user or area dump is output to disk if the response to the query in the message IDA0N45 is Y[ES]. If /MODIFY-TEST-OPTIONS USERDUMP-OPTIONS=*PARAMETERS(DUMP=*YES) was issued beforehand, the message IDA0N45 is not output.

(11) The file containing the user dump is cataloged under the user ID of the caller.

(12) see (10)

(13) The file containing the area dump is cataloged under the user ID of the caller.

The dumps can be edited for the interactive evaluation or printout with the utility routine DAMP (see “Diagnostics Handbook” [9]).

CHKEI – Check event item

General

Application area:	Eventing, see page 94
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

Macro description

The **CHKEI** macro communicates information to the calling program on the status of the queues associated with the specified event item.

Macro format and description of operands

```
CHKEI
---------------------------------------------------------------------------------------
 {  { EINAME=name                                }                                     }
 {  { EINAMAD=addr / (r) [,EINAMLN=length]       },SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL }
 {                                                                                     }
 {  EIID=addr / (r)                                                                    }

[,PARMOD=24 / 31]
[,MF=L / (E,..)]
```

EINAME=name
Specifies the name of the event item. The event item must already have been enabled before the **CHKEI** macro is issued.
The SCOPE operand must be specified for unique event item identification.

EINAMAD=
Specifies the event item. The event item is unique only if an additional SCOPE operand has been specified.

addr
Symbolic address of the field containing the name of the event item.

(r)
Register containing the address of the field.

EINAMLN=
Specifies the length in bytes of the name of the event item. The length must be at least 1 byte and must not exceed 54 bytes.
If the operand is omitted, the length attribute of the EINAMAD operand is assumed if EINAMAD=addr is specified; if EINAMAD=(r), the maximum length of 54 bytes is assumed.

length
Length of the event item name.

SCOPE=
Specifies the scope of eventing (participants using the event item).

LOCAL
The use of the event item is limited to the calling task.

GROUP
All the tasks with the same user ID as the calling task are participants.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.

The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP; the program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

EIID=
Specifies the ID of the event item. This ID is supplied to the user by the **ENAEI** macro. If the ID is used instead of the name of the event item, processing is speeded up. The event item is uniquely identified by the ID.

addr
Symbolic address of the field containing the ID.

(r)
Register containing the address of the field.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Return information and error flags

Register 1 contains the operand list address (during macro execution) and the number of pending **SOLSIG** or **POSSIG** macros (after macro execution).

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the CHKEI macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'28'	X'00'	Function executed: one or more SOLSIG requests in the queue. The number is specified in R1
X'2C'	X'00'	Function executed: one or more POSSIG requests in the queue. The number is specified in R1
X'30'	X'00'	Function executed: no requests in the queues
X'0C'	X'04'	No action taken: the event item generated by the system is not allocated to the calling task
X'10'	X'04'	No action taken: invalid operands were specified
X'14'	X'04'	No action taken: invalid name or invalid ID. There is no event item with the specified ID

For **examples**, see the sections “Eventing” (page 106) and “Contingency processes” (page 118) and the **SOLSIG** macro description (page 816).

CHKPRV – Check system privileges

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Type S, MF format **2**: standard/E/L/M/C/D form; see page 29

Macro description

In order to provide better protection for privileged system services against unauthorized access the SECOS software product allows privileges possessed by the holder of the TSOS user ID (system administration) in earlier versions to be assigned to various management facilities, each responsible for a subarea of the system administration. Each of these facilities possesses only the system privileges which it requires for its tasks. Details of how these privileges are distributed can be found in the “SECOS” manual [14].

Using the **CHKPRV** macro, users can check in their program whether the job running the program possesses one or more of these system privileges. 31 system privileges can be checked. The result of the check is entered in the standard header of the data area as a return code (see return code table following the operand description).

Macro format and description of operands

CHKPRV
PRIV=(priv[,priv]...) ,MF=S / E / L / C / D / M [,PARAM=addr / (r)] ,PREFIX=S / p ,MACID=RMC / macid

PRIV=
Specifies the system privilege to be checked.

(priv[,priv]...)
Descriptions of the privileges to be checked. If several privileges are specified, the return code “No error” is given if the job possesses at least one of these privileges.

The following overview lists the possible values for priv with the corresponding system privileges:

ACSADM	Alias catalog service administration
CUPRV001 : CUPRV008	flexibly assigned by system administration ((enables individual users to be granted individual rights))
FTACADM	FTAC administration
FTADM	File transfer administration
GUAADM	System global guard administration
HSMSADM	HSMS administration
HWMAINT	Hardware online maintenance
NETADM	Network administration
NOTIFADM	Notification service administration
OPERATING	BS2000 system operation
POSIXADM	POSIX user administration
PROPADM	Programming of administration procedures
PRSRVADM	SPOOL administration
SATFEVAL	SAT file evaluation
SATFMGMT	SAT file management
SECADM	Security administration
STDPROC	Using user commands
SUBSMGMT	Subsystem management
SWMONADM	Software monitor administration
TAPEADM	Tape administration
TAPEKEYADM	Encryption Key management for tapes
TSOS	TSOS privileges which are not assigned to any of the other management facilities specified here
USERADM	User administration
VMPRIV	Virtual machine administration
VM2ADM	VM2000 administration

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form, D form or M form of the macro and a MACID in the C form or M form (see section “S-type macros” on page 29).

Return information and error flags

Standard header:

0	0	b	b	a	a	a	a

The following return code relating to the macro CHKPRV is transferred in the standard header (bb=Subcode1, aaaa=Maincode):

X'bb'	X'aaaa'	Meaning
X'00'	X'0000'	Function successfully executed. The task possesses at least one of the specified privileges
X'00'	X'0002'	The task possesses none of the specified privileges
X'01'	X'0003'	Operand error: illegal specification for privileges
X'20'	X'00FF'	System error

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

The calling program is terminated when the following errors occur:

- The data area is not assigned to the caller.
- The data area is not aligned on a word boundary.
- The data area is protected against write access.

CHKSI – Check serialization item

General

Application area:	(Task) serialization; see page 91
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

CHKSI generates either a 24-bit or a 31-bit interface, depending on the specification. In the event of macro chaining, all macros chained must make use of the same interface (either 24-bit or 31-bit interface).

Macro description

This macro is used to check the status of a serialization item. The result of the check is stored in register 15 (see “Return information and error flags” below).

At **CHKSI** macro execution time, the specified serialization item must be present; implicit creation is not performed.

The CONTINU operand permits up to 255 **CHKSI** macros to be chained.

Macro format and description of operands

```
CHKSI

 ⎧ ⎧ SINAME=name                          ⎫                                                ⎫
 ⎨ ⎨ SINAMAD=addr / (r) [,SINAMLN=length] ⎬,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL        ⎬
 ⎪ ⎩                                      ⎭                                                ⎪
 ⎩ SIID=addr / (r)                                                                         ⎭

,CONTINU=NO / YES
[,PARMOD=24 / 31]
[,MF=L / (E,..)]
```

SINAME=name

Specifies the name of the serialization item. The SCOPE operand is required for unique serialization item identification.

SINAMAD=
Specifies the name of the serialization item. The serialization item is uniquely identified only if the SCOPE operand has also been specified.

addr
Symbolic address of the field containing the name.

(r)
Register containing the address.

SINAMLN=
Specifies the length in bytes of the serialization item name. The length must be at least 1 byte and not more than 54 bytes.
If the operand is missing, the length attribute of the SINAMAD operand is assumed if SINAMAD=addr is specified; if SINAMAD=(r), the maximum length (54) is assumed.

length
Length of the serialization item name in bytes.

SCOPE=
Specifies the scope of the serialization item (participants making use of the item).

LOCAL
The use of the serialization item is limited to the calling task.

GROUP
All the tasks with the same user ID as the caller are participants.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.

The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

SIID=
Specifies the ID of the serialization item. The ID is returned to the user by the **ENASI** macro. Use of this ID instead of the name of the serialization item increases processing speed. The ID is unique.

addr
Symbolic address of a 4-byte field containing the ID.

(r)
Register containing the address.

CONTINU=
Specifies whether up to 255 **CHKSI** macros are to be chained.

NO
This macro is the last (or only) macro of a sequence.

YES
Indicates that this macro is followed by another **CHKSI** macro.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Notes on the macro call using the list form (MF=L operand)

Only one macro call with MF=E need be issued for execution, irrespective of whether this call applies to a single request or to a series of requests. The operand list for a series of requests is generated by macro call chaining (MF=L) by means of the CONTINU operand.

Return information and error flags

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the CHKSI macro is transferred in register R15.

X'bb'	**X'aa'**	**Meaning**
X'28'	X'00'	All serialization items were checked. All specified serialization items are available, i.e. none of the serialization items is being used
X'2C'	X'00'	All serialization items were checked. All specified serialization items are being used by the calling program' s task, i.e. the access requests for these serialization items were satisfied
X'30'	X'00'	All serialization items were checked. One or more of the specified serialization items are being used by the calling program' s task; the remaining ones are available
X'34'	X'00'	All serialization items were checked. The calling program' s task is not using any of the specified serialization items, but one or more serialization items are being used by other tasks
X'38'	X'00'	All serialization items were checked. One or more of the specified serialization items are being used by the calling program' s task and by other tasks
X'10'	X'04'	The serialization items were not all checked. Invalid operands were specified: – Invalid address, i.e. address within a DSECT – Invalid length – Invalid name – SCOPE or CONTINU value undefined
X'14'	X'04'	The serialization items were not all checked. An invalid ID was specified
X'20'	X'04'	The serialization items were not all checked. There was at least one serialization item for which the calling program did not perform the enable function (see the ENASI macro)

CLCOM – Terminate intertask communication

General

Application areas:	Intertask communication (ITC); see page 76 Communication; see page 163
Macro type:	Type R; see page 28

Macro description

By means of **CLCOM**, the calling program terminates its participation in ITC (by deleting its ITC name from the list of participants).

Macro format and description of operands

CLCOM
NOKEEP / KEEP / (1)

NOKEEP
Specifies that the receive queue is dissolved. Any messages still in the queue can no longer be requested.

KEEP
Specifies that the receive queue is retained. Outstanding messages in the receive queue may still be requested after the **CLCOM** call has been issued.

(1)
Register R1 contains the start address of the operand field. This operand field has a length of 4 bytes and starts at a word boundary; its contents are as follows:

X'04000000'	corresponds to NOKEEP or
C'KEEP'	corresponds to KEEP

Functional description

By means of **CLCOM**, the calling program's task terminates its participation in intertask communication (ITC). Once a program has called **CLCOM** NOKEEP, it can neither send nor receive any more messages. Messages still in its receive queue are deleted.

If **CLCOM** KEEP is specified, the receive queue is not deleted. The program may still send messages and also request any that have already arrived. Once **CLCOM** KEEP is invoked, no new messages will be transferred. If there were no messages in the receive queue at the time **CLCOM** KEEP was called, then the system proceeds as if **CLCOM** NOKEEP had been specified.

The call **CLCOM** NOKEEP causes the system to delete the ITC name of the calling program from the list of participants. The ITC name can then be used again. If desired, the calling program may join ITC again (under any ITC name that has not already been assigned). When the last ITC participant issues the **CLCOM** NOKEEP call, ITC is terminated (the system deletes the ITC table).

Note

> If the NOKEEP operand is omitted (i.e. the default value is to apply), but a comment is written in this line, then the system executes the **CLCOM** KEEP function.

Return information and error flags

R15:

						a	a

A return code relating to the execution of the CLCOM macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	ITC participation is terminated, and the receive queue is dissolved
X'04'	Operand error. ITC participation continues
X'08'	The task of the calling program is not an ITC participant
X'0C'	There are still messages in the receive queue. ITC participation still has to be terminated (only in conjunction with KEEP)

CMD – Call command

General

Application areas:	Macro Command Language Processor macros; see page 45 Communication; see page 163
Macro type:	Type S, MF format **1**: standard/L/D/C/E form; see page 29

Macro description

The **CMD** macro enables a command or a list of commands to be called without exiting program mode. The **CMD** macro activates the macro command language processor (MLCP) and passes to it the command name and the specified command operands or a list of command names and the associated command operands. After the (last) command has been executed, the program continues.
Users can have a log of the command processing (e.g. system messages) transfered to SYSOUT and/or to an area of their program or to an S variable. The command return code can also be transfered to an area of the program.

The table 12 on page 315 sets out the commands which cannot be called using the **CMD** macro.

Macro format and description of operands

```
CMD
------------------------------------------------------------
                    ⎧ ,'oplist1'[, addr / (r)] ⎫
'command-name' [    ⎨                          ⎬ ]
                    ⎩ , , addr / (r)           ⎭

[,OPART2='oplist2'] [,OPART3='oplist3'] [,OPART4='oplist4']
[,OPART5='oplist5'] [,OPART6='oplist6'] [,OPART7='oplist7']
,LIST=NO / YES
[,CMDRC=addr]
,DIALOG=NO / YES
,SYSOUT=YES / NO
,SUBST=NO / JV / ALL
,ORIGIN=CMD / CURRENT
[,DTAVAR@=addr]
[,DTAVARL=length]
```

CMD (cont.)
,DTAEXT=NO / YES
[,MSGVAR@=addr]
[,MSGVARL=length]
,MSGEXT=NO / YES
,VER=1 / 2 / 3 / 4
,BUFMOD=SHORT / LONG
[,PARMOD=24 / 31]
[,MF=L / (E,..) / D / C]
,PREFIX=M / p

The keyword operands are listed in alphabetical order after the descriptions of the positional operands.

'command-name'
Name of the command which is to be called by **CMD**.
If the LIST=YES operand is specified, the 'command-name' string can contain a list of commands separated by semicolons. Each command is separated from its list of command operands by a blank.
If an error occurs while the commands in this list are being processed, the **CMD** macro is aborted at this point.
If the 'command-name' operand contains a list of commands, the 'oplist1' and OPARTx operands are omitted.

'oplist1'
oplist1 = op1,op2,op3,..... = list of the command operands.
The length of the list must be ≤ 248 characters. The list can be continued at any desired point using the operands OPART2, ..., OPART7. Each successive list is regarded as the continuation of the one immediately preceding it (without regard for the numbering 2-7). The operand may not be specified in conjunction with LIST=YES.

addr
addr = address of the receiving field for the SYSOUT listing. If “addr” is not specified, output will be to SYSOUT only. This is also the case if the length of the receiving field is zero. The receiving field must be aligned on a word boundary. Structure and maximum length of the receiving field depend on the value of the BUFMOD operand.
The first four bytes of every record in the SYSOUT listing that is copied into the receiving area represent a record length field (bytes 0-1 contain the record length, bytes 2-3 are reserved). The output text itself starts at byte 4 of each record. The output records are

written to the receiving area one after another until the area limit has been reached. When no more space is left in this area, any further output records are written to SYSOUT only (if SYSOUT=YES is specified).
A record may be truncated if it oversteps the area limit (return code X'OC').

Note

The SYSOUT output format for a command in interactive mode may differ from that in batch mode. This must be taken into account when defining the receiving area.

(r)
Register containing the address of the receiving field.

BUFMOD=
Defines the structure and maximum size of the receiving field for the SYSOUT listing.

SHORT
The receiving field may be up to 32 Kbytes long and has the following structure:

Byte 0-1: length l (hex.) of the receiving field in bytes (1 ≤ 2^{15}-1); l is to be specified by the user.

Byte 2-3: reserved, no entry

Byte 4-n: SYSOUT listing

LONG
May only be specified if the 31 bit interface of the macro is generated (PARMOD=31). This value may only be specified in conjunction with VER=2/3/4. The receiving field may be up to 2 Gb long and has the following structure:

Byte 0-3: length l (hex.) of the receiving field in bytes ($l \le 2^{31}-1$); l is to be specified by the user. The following values are to be observed for l:

`l=0:` The receiving field is ignored
`1 ≤ l ≤ 16:` Macro execution is aborted with the return code X'08'.

Byte 4-7: length l (hex.) of the user data in the receivingfield in bytes (including the 16-byte prefix) is entered by the system when a macro is executed. Default: length=16.

Byte 8-11: reserved; however, must be deleted when the CMD macro is called (binary zero or -1), otherwise execution is aborted with the error code X'08'.

Byte 12-15: reserved; used by the system

Byte 16-n: SYSOUT listing

Note

If the receiving field is not fully assigned, macro execution is aborted with the return code X'08'.
If the specified length `l` is exceeded, output is truncated as per BUFMOD=SHORT. **CMD** returns the return code X'0C'.

CMDRC=
Specifies the symbolic address of a 9-byte field in the following format to which the command return code of the command processed by the **CMD** macro is written.

Byte 0:	Subcode2 in assembler format X'nn'
Byte 1:	Subcode1 in assembler format X'nn'
Byte 2-8:	Maincode in assembler format CL7

If a list of commands is passed in the 'command-name' operand (only possible in conjunction with LIST=YES), the specified field contains the return code of the last command to be passed to MCLP.
The operand may be specified only in conjunction with VER=3/4.

addr
Symbolic address of the field to which the return code is to be written. Default setting: 0, i.e. no return code is transferred.

DIALOG=
This indicates whether an error or help dialog is to be conducted if syntax errors are detected.

NO
No error dialog is conducted.

YES
Error dialog is to be conducted, if the terminal type allows this.

DTAEXT=
This determines whether or not the S variable specified in DTAVAR@ should be extended to include the contents of the variables generated by OPS.
This operand may only be specified in conjunction with VER=4 and specifications for DTAVAR@ and DTAVARL.

NO
The S variable should not be extended.
The (old) contents of S variables are replaced by the contents of the OPS variables.

YES
The S variable is extended by adding the contents of the OPS variables. If the S variable cannot be extended, the (old) contents are deleted before saving.

DTAVAR@=
specifies the symbolic address of an area containing the name of a composite S variable. The contents of all variables generated by OPS should be stored in this S variable.
If the S variable was not declared before the macro is called execution is terminated with the return code X'10'.
This operand may only be specified in conjunction with VER=4.

addr
Symbolic address of the area containing the name of S variables. This can be identical to the address specified in the MSGVAR@ operand so that all variables generated by MIP or OPS are saved in a single area.

DTAVARL=
This specifies the length of the area addressed in DTAVAR@.
This operand may only be specified in conjunction with VER=4.

length
length = length of the field to be specified in bytes.

LIST=
Enables a list of commands and their associated operands to be specified for the 'command-name' operand.

NO
The 'command-name' operand consists of a single command.

YES
The 'command-name' operand contains a list of commands separated by semicolons. This value may be specified only in conjunction with VER=3/4.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX (p = 1 letter) can be specified for MF=C or MF=D, as shown in the macro format.

MSGEXT=
This determines whether or not the S variable specified in MSGVAR@ should be extended to include the contents of the variables generated by MIP.
This operand may only be specified in conjunction with VER=4 and specifications for MSGVAR@ and MSGVARL.

NO
The S variable should not be extended.
The (old) contents of S variables are replaced by the contents of the MIP variables.

YES
The S variable is extended by adding the contents of the MIP variables. If the S variable cannot be extended, the (old) contents are deleted before saving.

MSGVAR@=
specifies the symbolic address of an area containing the name of a composite S variable. The contents of all variables generated by MIP should be stored in this S variable.
If the S variable was not declared before the macro is called execution is terminated with the return code X'10'.
This operand may only be specified in conjunction with VER=4.

addr
Symbolic address of the area containing the name of S variables. This can be identical to the address specified in the DTAVAR@ operand so that all variables generated by MIP or OPS are saved in a single area.

MSGVARL=
This specifies the length of the area addressed in MSGVAR@.
This operand may only be specified in conjunction with VER=4.

length
length = length of the field to be specified in bytes.

OPARTx=
Allows continuation of the operand list.
x = a member of the set (2, 3, .., 7).

'oplistx'
oplistx = op_i,op_j,op_k,.... = (continuation) list of command operands;
x = a member of the set (2,3,...,7).
Length: oplistx = 1..248 characters.
The operand may not be specified in conjunction with LIST=YES.

ORIGIN=
Specifies the origin of the command and must be used when checking whether the command is permitted.
This operand may only be specified in conjunction with VER=4 and MF=D or MF=C.

CMD
The command input must be checked with the CMD-ALLOWED attribute from the syntax file.

CURRENT
In addition to the CMD-ALLOWED attribute, the SDF also checks whether the command is permitted in the current mode, i.e. the mode in which the program was started. This mode may be: interactive, interactive procedure, batch or batch procedure.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb). This value may only be specified in conjunction with VER=1.

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

SUBST=
Specifies what substitutions are to take place in the command input.

NO
No substitution is to take place in the command input.

JV
All job variables in the command input are to be replaced by their contents. This value may only be specified in conjunction with VER=4.

ALL
Depending on the program environment, the entire command input is to be replaced, i.e. all S variables, job variables and SYSFILE procedure parameters in the command input are to be replaced in the specified order.
Substitution of SYSFILE procedure parameters is not meaningful for S procedures.
S variables are replaced only if they are known in the current program environment.
This value may only be specified in conjunction with VER=4.

SYSOUT=
This indicates whether the log is also to be output to SYSOUT.

YES
Output also to SYSOUT.

NO
No output to SYSOUT.
The receiving area “addr” must be specified in this case.

VER=1 / 2 / 3 / 4

Only important in conjunction with MF=C/D and is otherwise ignored. The operand defines the version of the operand list for which a DSECT is to be generated.
A DSECT is always generated.
Compatibility with other operands: see table below.

Compatibility with other operands: see table below.

		VER=1	VER=2	VER=3	VER=4
PARMOD	= 24	x			
	= 31	x	x	x	x
BUFMOD	= SHORT	x	x	x	x
	= LONG		x	x	x
LIST	= NO	x	x	x	x
	= YES			x	x
CMDRC				x	x
SUBST	= NO	x	x	x	x
	= JV				x
	= ALL				x
ORIGIN					x
DTA...					x
MSG...					x
other operands		x	x	x	x

Table 11: compatibility of the VER operand with other operands (CMD)

x means that the operands involved may be combined.

Notes on the macro call

- The specification DIALOG=YES is ignored in expert mode (MODIFY-SDF-OPTIONS GUIDANCE=EXPERT; see the manual “Commands” [19]). This results in the return code X'10' (instead of X'14').
- If DIALOG=NO is specified SDF delivers the following return code if the help request is used in the CMD input:
  - “?”: The return code X'10' is delivered.
  - “<operation?>”: (? as part of <operation>) : The return code X'14' is delivered.
  - “<operation> ?” : If <operation> is valid, the return code X'10' is delivered. If <operation> is invalid, the return code X'14' is delivered.

  In both modes (interactive or batch), an unclear command is returned for all values of the DIALOG operand with the X'14' return code.
- Processing of the command list (if LIST=YES) is terminated if one of the specified commands causes an error (syntax error, memory error,..). This type of error does not result in SPIN-OFF.
- The data area generated with MF=C/D must correspond to the data area used with MF=L.
- If the **CMD** parameter list was initialized with the standard form or with MF=L, the `<PREFIX>CLPLNTH` field contains the following information:
  - the length of the entire predefined parameter list of the **CMD** macro in the first halfword (i.e. 2 bytes, starting at byte 0).
  - the length of the processed command input in the second halfword (i.e. 2 bytes, starting at byte 2). If processing of the **CMD** macro was aborted due to an error, this halfword indicates the point at which processing was aborted.
- Structured **CMD** macro output cannot be directed to the S variable stream SYSINF (as with EXECUTE-CMD command).
- If the **CMD** macro is to direct its output to an S variable generated by OPS, then:
  - the START-PROGRAM command cannot be issued by the **CMD** macro;
  - it is not possible to chain a number of commands in a single input record ('command name' and LIST=YES operands).

– Not all the BS2000 commands can be called via the **CMD** macro, see the following table.

Command	Function	Manual
ADD-CJC-ACTION	Submit CJC command sequence (job variables)	[19]
BEGIN-BLOCK	Initiate command block	[21]
BEGIN-PARAMETER-DECLARATION	Initiate declaration of procedure parameters in procedure header	[19],[21]
BEGIN-PROCEDURE	Define procedure file attributes	[19]
BREAK	Request command mode	[34]
CANCEL-PROCEDURE	Terminate (execution of a) procedure	[19]
CANCEL-PROGRAM	Terminate program run	[19]
COPY-SYSTEM-FILE	Copy system files	[19]
CYCLE	Terminate loop execution	[21]
DELON	Delete ON command	[34]
ENDON	Terminate ON statement sequence	[34]
ELSE	Initiate ELSE branch in IF block	[19],[21]
ELSE-IF	Initiate alternative branch in IF block	[21]
END-BLOCK	Terminate a command block	[21]
END-CJC-ACTION	Terminate CJC command sequence (job variables)	[19]
END-FOR	Terminate a FOR block	[21]
END-IF	Terminate an IF block	[19],[21]
END-PARAMETER-DECLARATION	Terminate a procedure parameter declaration	[19],[21]
END-PROCEDURE	Terminate procedure file	[19]
END-WHILE	Terminate a WHILE block	[21]
ENDP	Terminate procedure file	[34]
EOF	Mark the end of the SYSDTA file	[19]
ESCAPE	Interrupt procedure run	[34]
EXIT-BLOCK	Interrupt processing of a command block	[21]
FOR	Initiate FOR block	[21]
GOTO	Branch to tag	[19],[21]
HOLD-PROCEDURE	Interrupt procedure execution and allow command input via the data display terminal	[19]
HOLD-PROGRAM	Interrupt program execution and allow command input via the data display terminal	[19]

Table 12: Commands that cannot be called via the CMD macro (Teil 1 von 2)

Command	Function	Manual
IF	Initiate a block	[19],[21]
IF-BLOCK-ERROR	Initiate a block error handling routine	[19],[21]
IF-CMD-ERROR	Initiate a command error handling routine	[21]
LOGON	Initiate job	[34]
MODIFY-ACCOUNTING-PARAMETERS	Specify accounting records and record extensions for the accounting file	[19]
MODIFY-JV-CONDITIONALLY	Modify the value of a job variable and branch to branch destination	[19],[22]
ON	Conditionally execute a command sequence	[34]
PROCEDURE	Specify procedure file attributes	[34]
REMOVE-CJC-ACTION	Cancel effect of ADD-CJC-ACTION command (job variables)	[19]
REPEAT	Initiate REPEAT block	[21]
RESTART-PROGRAM	Start a program at its checkpoint	[19]
RESUME-PROCEDURE	Continue execution of interrupted procedure	[19]
SELECT-PRODUCT-VERSION	Select a product version	[19]
SET-JOB-STEP	Terminate spin-off	[19]
SET-LOGON-PARAMETERS	Initiate an interactive or batch job	[19]
SHOW-ACCOUNTING-STATUS	Display information about accounting system	[19]
SKIP-COMMANDS	Conditional or unconditional branch	[19]
SKIPJV	Branch conditional upon job variables	[34]
SKIPUS	Branch conditional upon user switches	[34]
START-ACCOUNTING	Deactivate accounting system	[19]
STEP	Terminate spin-off	[34]
STOP-ACCOUNTING	Deactivate accounting system	[19]
UNTIL	Terminate REPEAT block	[21]
WAIT-EVENT	Specify conditional wait time (batch job	[19],[22]
WHEN	Set conditional halt for batch job (via user switch)	[34]
WHILE	Initiate WHILE block	[21]

Table 12: Commands that cannot be called via the CMD macro (Teil 2 von 2)

– When the following commands are called via the **CMD** macro, the calling program is unloaded:

Command	Function	Manual
ABEND	Terminate the current job	[34]
CALL	Call procedure	[34]
CALL-PROCEDURE	Call procedure	[19]
DO	Call procedure	[34]
EXECUTE	Load and start a module	[34]
EXIT-JOB	Terminate job	[19]
HELP-SDF	Help information on calling SDF commands	[19]
LOAD	Load a module	[34]
LOAD-EXECUTABLE-PROGRAM	Load a module	[19]
LOAD-PROGRAM	Load a module	[19]
LOGOFF	Terminate job	[34]
START-EXECUTABLE-PROGRAM	Link, load and start a module	[19]
START-PROGRAM	Link, load and start a module	[19]

Table 13: Commands that cause the calling program to be unloaded

The calling program is also unloaded if the called command is implemented by a command procedure. This is, for example, the case for all EDIT commands (see the “Commands” manual [19]). However, self-defined commands of this type can also exist by means of SDF-A (see the “SDF-A” manual [20]).

To prevent unloading, commands which are initialized via a command procedure and the CALL-PROCEDURE, CALL and DO commands can be called indirectly using the SDF-P command INCLUDE-CMD (see the “SDF-P” manual [21]). The command which is actually to be executed is called as an operand of INCLUDE-CMD.
INCLUDE-CMD interrupts the calling program, executes the command specified as an operand (and thus the associated command procedure) and then returns to the program. The called command procedure for its part may not, however, execute any command which leads to the program being unloaded.

A consequence is that in the case of commands which terminate the calling program, the SYSOUT listing will not be copied into the “addr” receiving area. Any STXIT routine which is defined for the event class “program termination” will be activated.
Any job variable which monitors the job will be set to “$T”.

Return information and error flags

R15:

						a	a

A return code relating to the execution of the CMD macro is specified in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Function executed successfully
X'04'	The function was aborted due to insufficient memory
X'08'	The function was aborted due to an error in the operand list (address area)
X'0C'	The function was aborted: the receiving area is too short. The last output record placed in the receiving area was truncated
X'10'	The function was aborted due to a macro/command error (the command returned an error to the MCLP)
X'14'	The function was aborted due to an invalid command in the operand list
X'24'	The function was aborted due to an error during substitution of the command input

Additionally, if PARMOD=31:

Standard header:

c	c	b	b	a	a	a	a

The following return code regarding execution of the CMD macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

cc	bb	aaaa	Meaning
00	00	0000	The function was executed successfully
00	01	0008	The function was aborted due to an operand error
00	20	0004	The function was aborted due to an internal error
00	40	0010	The function was aborted due to a macro/command error
00	40	0014	The function was aborted due to an invalid command in the operand list
00	40	0024	The function was aborted due to an error during substitution of the command input
02	00	000C	The function was aborted because the receiving area is too short. The last output record placed in the receiving area was truncated

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Layout of the DSECT for VER=4

```
           CMD MF=D,PARMOD=31,VER=4
1          #INTF REFTYPE=REQUEST,INTNAME=CMD,INTCOMP=4
1          DS    0F
1          MFCHK DMACID=CLP,PREFIX=M,MACID=CLP,MF=D,DNAME=DMCLP
2 MDMCLP   DSECT ,
2                *,##### PREFIX=M, MACID=CLP #####
1 MCLPSTRT DS    0F
1          FHDR  MF=(C,MCLP)
2          DS    0A
2 MCLPFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 MCLPIFID DS    0A              0   INTERFACE IDENTIFIER
2 MCLPFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 MCLPFCT  DS    AL1             2   FUNCTION NUMBER
2 MCLPFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 MCLPRET  DS    0A              4   GENERAL RETURN CODE
2 *
2 * GENERAL_RETURN_CODE CLEARED (X'00000000') MEANS
2 * REQUEST SUCCESSFUL PROCESSED AND NO ADDITIONAL INFORMATION
2 *
2 MCLPSRET DS    0AL2            4   SUB RETURN CODE
2 MCLPSR2  DS    AL1             4   SUB RETURN CODE 2
2 * ALWAYS CLEARED (X'00') IF MAIN_RETURN_CODE IS X'FFFF'
2 * Standard subcode2 values as defined by convention:
2 MCLPR2OK EQU   X'00'               All correct, no additional info
2 MCLPR2NA EQU   X'01'               Successful, no action was necessary
2 MCLPR2WA EQU   X'02'               Warning, particular situation
2 MCLPSR1  DS    AL1             5   SUB RETURN CODE 1
2 *
2 * GENERAL INDICATION OF ERROR CLASSES
2 *
2 * CLASS A    X'00'           FUNCTION WAS SUCCESSFULLY PROCESSED
2 * CLASS B    X'01' - X'1F'   PARAMETER SYNTAX ERROR
2 * CLASS C    X'20'           INTERNAL ERROR IN CALLED FUNCTION
2 * CLASS D    X'40' - X'7F'   NO CLASS SPECIFIC REACTION POSSIBLE
2 * CLASS E    X'80' - X'82'   WAIT AND RETRY
2 *
2 MCLPRFSP EQU   X'00'               FUNCTION SUCCESSFULLY PROCESSED
2 MCLPRPER EQU   X'01'               PARAMETER SYNTAX ERROR
2 *  3 GLOBALLY DEFINED ISL ERROR CODES IN CLASS X'01' - X'1F'
2 MCLPRFNS EQU   X'01'               CALLED FUNCTION NOT SUPPORTED
```

```
2 MCLPRFNA EQU   X'02'               CALLED FUNCTION NOT AVAILABLE
2 MCLPRVNA EQU   X'03'               INTERFACE VERSION NOT SUPPORTED
2 *
2 MCLPRAER EQU   X'04'               ALIGNMENT ERROR
2 MCLPRIER EQU   X'20'               INTERNAL ERROR
2 MCLPRCAR EQU   X'40'               CORRECT AND RETRY
2 *  2 GLOBALLY DEFINED ISL ERROR CODES IN CLASS X'40' - X'7F'
2 MCLPRECR EQU   X'41'               SUBSYSTEM (SS) MUST BE CREATED
2 *                                  EXPLICITELY BY CREATE-SS
2 MCLPRECN EQU   X'42'               SS MUST BE EXPLICITELY CONNECTED
2 *
2 MCLPRWAR EQU   X'80'               WAIT FOR A SHORT TIME AND RETRY
2 MCLPRWLR EQU   X'81'                    "     LONG        "
2 MCLPRWUR EQU   X'82'               WAIT TIME IS UNCALCULABLY LONG
2 *                                  BUT RETRY IS POSSIBLE
2 *  2 GLOBALLY DEFINED ISL ERROR CODES IN CLASS X'80' - X'82'
2 MCLPRTNA EQU   X'81'               SS TEMPORARILY NOT AVAILABLE
2 MCLPRDH  EQU   X'82'               SS IN DELETE / HOLD
2 *
2 MCLPMRET DS    0AL2            6   MAIN RETURN CODE
2 MCLPMR2  DS    AL1             6   MAIN RETURN CODE 2
2 MCLPMR1  DS    AL1             7   MAIN RETURN CODE 1
2 *
2 * SPECIAL LAYOUT OF LINKAGE_MAIN_RETURN_CODE (YYYY IN X'00XXYYYY')
2 *
2 MCLPRLNK EQU   X'FFFF'             LINKAGE ERROR / REQ. NOT PROCESSED
2 MCLPFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1 MCLPFLAG DS    X                       Flag bits
1 *              Flag bits:.0......    : UNUSED
1 *                        0.......    : reserved for TPR usage
1 MCLPDIA  EQU   X'20'     ..1.....    : DIALOG possible (w. SDF only)
1 MCLPNSYS EQU   X'10'     ...1....    : no SYSOUT logging
1 MCLPLIST EQU   X'08'     ....1...    : LIST of commands
1 MCLPNOUT EQU   X'04'     .....10.    : no output buffer
1 MCLPLONG EQU   X'02'     .....01.    : long output buffer
1 *                        .....00.    : short (old) output buffer
1 MCLPREG  EQU   X'01'     .......1     buffer @ given via a register/
1 MCLPADDR EQU   X'00'     .......0     buffer @ given directly
1 MCLPNSRG EQU   X'11'     ...1...1     no SYSOUT, buff.@ in a reg.
1 MCLPNSAD EQU   X'10'     ...1...0     no SYSOUT, buff.@ given directly
1 MCLPFLA3 DS    XL1                    FLAG 3
1 MCLPSJV  EQU   X'80'     10......     substitute jv only
1 MCLPSALL EQU   X'40'     01......     substitute all
1 MCLPDATE EQU   X'20'     ..1.....     data var buf extend
1 MCLPMSGE EQU   X'10'     ...1....     msg var buf extend
1 MCLPCUOR EQU   X'04'     .....1..     origin=current
1 MCLPUNUS DS    XL1                    reserved (unused)
```

```
1 MCLPROUT DS    XL1                     reg.# if buffer @ in reg.
1 MCLPOUT  DS    A                       buffer @ or 0
1 MCLPCMD@ DS    A                       @ of command (-> V-field)
1 MCLPRC@  DS    A                       @ return code of command
1 MCLPTPR2 DS    A               reserved for TPR usage.
1 MCLPDAV@ DS    A                       @ of var name for data
1 MCLPMSV@ DS    A                       @ of var name for message
1 MCLPDAVL DS    H                       length of var name
1 MCLPMSVL DS    H                       length of var name
1 MCLPTPR3 DS    A               reserved for TPR usage.
1 MCLPHLN  EQU   *-MCLPSTRT            CMD p.l. length
1 MCLPLNTH EQU   *       command's V-field when CLPCMD@ points here
1 MCLPCMD  EQU   *+4     cmd start when MCLPCMD@ points to MCLPLNTH
```

Example

With the **CMD** macro, the SHOW-JOB-STATUS command is run in two variants. The program is executed in 31-bit addressing mode below the 16-Mb boundary.

```
CMD      START
         PRINT NOGEN
CMD      AMODE ANY
         BALR  3,0
         USING *,3
         CMD   MF=(E,LFORMAD1),PARMOD=31  ——————————————————————————————  (1)
         CMD   MF=(E,LFORMAD2),PARMOD=31  ——————————————————————————————  (2)
         MVC   MESSAGE(4),PROTCONT
         MVC   MESSTXT,PROTCONT+4
         WROUT MESSAGE,END,PARMOD=31
END      TERM
LFORMAD1 CMD   'SHOW-JOB-STA','INF=(*STD,*PROGRAM)',MF=L,PARMOD=31 ————  (3)
LFORMAD2 CMD   'SHOW-JOB-STA','INF=*STD',PROT,SYSOUT=NO,MF=L,PARMOD=31    (4)
         DS    0F
PROT     DC    Y(PROTEND-PROT) ————————————————————————————————————————  (5)
         DC    X'4040'
PROTCONT DS    CL2500
PROTEND  EQU   *
MESSAGE  DC    Y(ENDMESS-MESSAGE)      Record length
         DS    CL2                     Reserved
         DC    X'01'                   Print feed control character
MESSTXT  DS    CL255                   Contents
ENDMESS  EQU   *
         END
```

(1) The macro call is split into the instruction part and data area (see page 29). At this point, only the instruction part (SVC) is given, with a reference to the data area LFORMADR1 in the data section of the program.

(2) The macro is called with a reference to the data area LFORMADR2.

(3) The SHOW-JOB-STATUS command is called with the INF=(*STD,*PROGRAM) operand. Output is sent to the display terminal.

(4) The SHOW-JOB-STATUS command is called with the INF=*STD operand. The information is output to the `PROT` area.

(5) The command output area starts on a word boundary, with the area length being entered in the first two bytes.

Runtime log

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,cmd), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,cmd))
%  ASS6011 ASSEMBLY TIME: 344 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 80 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=cmd
%  BLS0523 ELEMENT 'CMD', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'CMD', VERSION ' ' OF '2<date> <time>' LOADED
NAME      TSN TYPE       PRI      CPU-USED CPU-MAX ACCOUNT# ───────────  (6)
MACTEST  2QSE 3 DIALOG1  0 210     58.7710    9000 89002
NAME      TSN TYPE        SIZE CURR-CMD
MACTEST  2QSE 3 DIALOG1      1 START-EXECUTABLE-PROGRAM
              PROG::2OSG:$QM212.MACEXMP.LIB(CMD,@,L)
NAME      TSN TYPE       PRI      CPU-USED CPU-MAX ACCOUNT# ───────────  (7)
MACTEST  2QSE 3 DIALOG1  0 210     58.7856    9000 89002
```

(6) Output of the first **CMD** call direct to the display terminal.

(7) The output area `PROT` of the second **CMD** call is displayed using **WROUT**.

CONTXT – Access process data

General

Application areas:	Contingency processing; see page 110 STXIT processing; see page 131
Macro type:	S-Typ, MF-Format **1**: standard/L/E form; see page 29

If a basic process or a contingency process is interrupted by a contingency process, the contents of its registers and of the program counter are stored in the PCB (process control block).

Macro description

The **CONTXT** macro gives a contingency process access to the context (PCB, Process Control Block) of an interrupted process.

CONTXT supports all current BS2000 servers. The interrupted process can therefore exist in /390 code with a /390 context (servers with /390 architecture) or in x86 code with an x86 context (servers with x86 architecture). Since there are differences between contexts, the LAYOUT operand is used to distinguish them.

If LAYOUT=COMPATIBLE (default), the complete context (register and PC) are read for an interrupted /390 process. The whole context can be modified and written.
For an interrupted x86 process, the equivalent parts of the context are mapped to the relevant areas of the layout. The context is therefore not output in its entirety. It can also only be written in parts.

If LAYOUT=FCONTXT, the complete context of an interrupted x86 process is read. ILC is formed. CC and PM are meaningless.
The complete context of an interrupted /390 process is mapped to the relevant areas of the layout.
The whole context of a process can be modified and written. CC, ILC and PM cannot be written for x86 processes.

See also “PCB accesses in the /390 process mode” on page 338.

If x86 code is integrated or dynamically loaded (e.g. dynamic loading of the ported product SORT) in user applications (/390-Code) both the context of x86 code and the context of /390 code are visible at this interface, depending on the mode in which the interruption occurred. Programs which run completely in /390 mode are not affected.

The following differences in the context can be seen when a x86 program is interrupted and the context of the interrupted (x86) process is viewed (`CONTXT SAVE=...,LAYOUT=FCONTXT`):

- the context contains the /390-equivalent parts (register, PC, floating-point register, etc.) which can also be addressed via the DSECT and
- an HSI-dependent area which can be read or written in block form

Macro format and description of operands

```
CONTXT
--------------------------------------------------------------

[ { SAVE=addr / (r)                          } ]
  { STACKR=(x1, x2, ...),OWNR=(y1, y2,...)   }

[, { SAVACR=addr / (r)                            } ]
   { STKACR=(x1, x2, ...),OWNACR=(y1, y2,...)     }

,FPR=NO / YES
,PROCESS=MAIN / LAST
,FUNCT=READ / WRITE
,LAYOUT=COMPATIBLE / FCONTXT
,LAYOUTF=DSECT[,PREFIX=p]
[,ILC=addr / (r)]
[,CC=addr / (r)]
[,PM=addr / (r)]
[,PMODE=addr / (r)]
[,MODE=addr / (r)]
[,ASCMOD=addr / (r)]
[,PRGCODE=addr / (r)]
[,PRGCODL=addr / (r)]
[,PARMOD=24 / 31]
[,MF=L / (E,..)]
```

The operands ILC, CC, PM are meaningless for servers with x86 architecture.

SAVE=
Describes the address of a field used for exchanging data with the PCB of the specified process (PROCESS operand). The structure and contents of this field depend on the LAYOUT operand.

– *LAYOUT=COMPATIBLE (preset value)*
  The data exchange field has the same structure as in the previous CONTXT interface in /390 mode.
  Field length = 68 bytes; data must be aligned on word boundaries.
  Field structure and assignments:

```
Byte  0 - byte 63: Register R0 through R15 of the /390 PCB
                   equivalent registers of the X86 PCB
Byte 64 b- byte 67: program counter (PC)
```

– *LAYOUT=FCONTXT*
  To support x86 mode the data exchange field has a structure of its own. In addition to the /390 PCB it also contains the entire x86 PCB and consequently also the floating-point registers. They are passed between the data exchange field and the PCB of the specified process, regardless of the value of the FPR operand.

  If, however, the process whose PCB is being accessed is running in /390 mode, then a limited amount of data is passed. Only the areas of the data exchange field that represent SPARC registers for which there are /390 equivalents are read or supplied with data.

  The field length of the data exchange field can be determined dynamically with the **STXIT** macro**,** CONTXTL operand. Validation of the data exchange field is, however, always implemented in the length used. The data exchange field must be aligned to doubleword boundary.

  A DSECT that describes the new structure of the data exchange field can be created by calling the **CONTXT** macro with the operand LAYOUTF=DSECT. With this DSECT, it is possible to symbolically address the individual subfields of the data exchange field.

If FUNCT=READ applies, the context is transferred from the PCB of the specified process to the specified field.

If FUNCT=WRITE applies, the content of the specified field is transferred to the context of the specified process. The program counter (PC / NIA = Next Instruction Address) can only be written if it points to a /390 module or if LAYOUT= FCONTXT has been specified.

addr
Symbolic address (name) of the area for data exchange.

(r)
r = register containing the address value “addr”

STACKR=
Designates a series of selected registers (including the program counter) of the specified process for data exchange. For x86 contexts, only the general-purpose registers are supported as being equivalent to the /390 registers R0 through R15. the content of the specified field is transferred to the context of the specified process. The program counter (PC / NIA = Next Instruction Address) can only be written if it points to a /390 module or if LAYOUT= FCONTXT has been specified. There are no such restrictions when reading the program counter.

Note
The same number of operand values must be specified for the STACKR and OWNR operands. The specifications enclosed in parentheses are paired for data transfer. Thus data transfer is effected between x1 and y1, between x2 and y2, between x3 and y3, etc.

(x1,x2,...)
x1,x2 = members of the set (0,1,2,.....,15,PC).
The numbers 0, ...,15 stand for the general registers, and PC for the program counter. In the case of the program counter, only the address of the next instruction is transferred.

OWNR=
Identifies a sequence of selected registers of the calling (contingency) process, which are to be used to receive the values read, or to contain the values to be written (see also the note on STACKR).

(y1,y2,...)
y1,y2 = members of the set (0,1,2,.....,15,PC).
The numbers 0, ...,15 stand for the general registers and PC stands for the program counter.
When STACKR and OWNR are used, it is important to note that the macro destroys the contents of registers R1 and R15 (register R1: address of the macro operand list, register R15: return information). Thus, it is not possible to “write” the contents from register R1 and to “read” the contents into register R15.

SAVACR=
Identifies the address of a field for data exchange with the PCB of the specified process (PROCESS operand).
Field length = 64 bytes; the field must be aligned on a word boundary.
If FUNCT=READ, the contents of the access registers AR0 through AR15 are transferred from the PCB of the specified process into the specified field.
If FUNCT=WRITE, the contents of the specified field are transferred into the access register of the specified process according to the above assignment.

addr
Symbolic address (name) of the field for data exchange.

(r)
r = register containing the address value “addr”.

STKACR=
Identifies a sequence of selected access registers of the specified process for use in data exchange.

Note
The same number of operand values must be specified for the STKACR and OWNACR operands. The specifications enclosed in parentheses are paired for data transfer. Thus data transfer is effected between x1 and y1, between x2 and y2, between x3 and y3, etc.

(x1,x2,...)
x1,x2 = members of the set (0,1,2,.....,15).
The numbers 0, ...,15 stand for the access registers.

OWNACR=
Identifies a sequence of access registers of the calling (contingency) process, which are to be used to receive the values read, or to contain the values to be written; (see also the note on STKACR).

(y1,y2,...)
y1,y2 = members of the set (0,1,2,.....,15).
The numbers 0, ...,15 stand for the access registers.

FPR=
Specifies whether or not the contents of the floating-point registers are to be transferred. This operand is only significant when used with LAYOUT=COMPATIBLE (preset value).
If one of the participating processes is running in x86 mode, then only floating-point registers of the process may participate in the transfer that represent the equivalent /390 floating-point registers.

NO
The contents of the floating-point registers are not to be transferred.

YES
If FUNCT=READ is specified, the contents of the floating-point registers of the specified process are transferred to the floating-point registers of the calling (contingency) process.
If FUNCT=WRITE is specified, the contents of the floating-point registers of the calling (contingency) process are transferred to the floating-point registers of the specified process.

If LAYOUT=FCONTXT is specified, there is no direct transfer of the contents of the floating-point registers of the calling process to the floating-point registers of the specified process or vice versa.
Instead, the floating-point registers are passed using the data exchange field that is specified by the SAVE operand. The FPR operand specification is ignored.

All the x86 mode floating-point registers are passed, regardless of the run mode of the calling process, if the specified process is running in x86 mode.

PROCESS=
Specifies the process which is to be accessed.

MAIN
The basic process is accessed, even if it was not the one directly interrupted by the calling (contingency) process.

LAST
The process that was interrupted by the calling process is accessed. This can be the basic process or another contingency process.

FUNCT=
Specifies whether a read or write access is required.
See also “PCB accesses in the /390 process mode” on page 338.

READ
The contents of the specified registers and, if required, the program counter and the complete x86 context (if LAYOUT=FCONTEXT on servers with x86 architecture), are read from the PCB of the specified process, to the specified fields.

WRITE
The specified registers and, if required, the program counter and the complete x86 context (if LAYOUT=FCONTEXT on servers with x86 architecture) of the specified process are overwritten by the values specified by the calling process.
Write access is only possible when the storage key in the PCB of the calling process matches the storage key in the PCB of the specified process.

/309 mode: The contents of the fields described with ILC/CC/PM/ASCMOD aretransferred to the PCB of the specified process (interrupted process or basic process). The information is to be stored in the corresponding bits.

LAYOUT=
Specifies the scope and structure of the data exchange field whose address is specified in the SAVE operand. This also controls the “write NIA” function via the STACKR=(PC) operand, see page 327.

COMPATIBLE
Preset value: the scope and layout of the /390 context is expected. The floating-point registers will only be passed if FPR=YES is also specified.

FCONTXT
The scope and layout of the x86 context is expected. For x86 contexts the floating-point registers are transferred via the data exchange field, regardless of the FPR operand. The length of the SAVE area to be provided by the user can be determined dynamically using the **STXIT** macro, CONTXTL operand.

The actual length required depends on the process mode. For information on the length and the layout of the relevant data exchange field see “Layout of the DSECT” on page 333.

LAYOUTF=DSECT
Triggers the creation of a DSECT for the data exchange field as in the operand LAYOUT=FCONTXT . The address of the data exchange field is specified in the SAVE operand when the action is called.
When this operand is specified, the only other operand evaluated is the PREFIX operand.

PREFIX=p
Specifies a letter which determines the first character of the field name and of the equate. This operand is only taken into account if LAYOUTF= DSECT is also specified.

ILC, CC, PM=
These operands are meaningless to a x86 PCB. They cannot be stored in a x86 PCB.

ILC=
Describes the address of a field for the instruction length code (in PCR format).
Field length = 1 byte. Entry in bits 0-1.

addr
Symbolic address (name) of the field for the instruction length code.

(r)
Register containing the address value “addr”.

CC=
Describes the address of a field for the condition code (in PCR format).
Field length = 1 byte. Entry in bits 2-3.

addr
Symbolic address (name) of the field for the condition code.

(r)
Register containing the address value “addr”.

PM=
Describes the address of a field for the program mask (in PCR format).
Field length = 1 byte. Entry in bits 4-7.

addr
Symbolic address (name) of the field for the program mask.

(r)
Register containing the address value “addr”.

PMODE=
Describes the address of a field containing the processor mode.
Field length = 1 byte.
The processor mode can be read or written on servers with x86 architecture. It can only be read on other BS2000 servers.
The following processor modes mean:

X'00': /390 mode (native on servers with /390 architecture or under /390 firmware on servers with x86 architecture)
X'01': x86 mode native

addr
Symbolic address (name) of the field containing the processor mode

(r)
r = register with the address value of addr.

MODE=
Defines the address of a 1-byte field which indicates the current addressing mode.
This is used for inquiring about or modifying the addressing mode used by the specified process.
The entry in this field has the following meaning:
X'00': 24-bit addressing mode
X'01': 31-bit addressing mode

If FUNCT=READ, the addressing mode is fetched from the PCB and entered in the specified field.
If FUNCT=WRITE, the specified addressing mode is entered in the PCB. When modifying the addressing mode, the user has to make sure that the context is compatible with the new addressing mode (24-bit or 31-bit).

addr
Symbolic address (name) of the field for the addressing mode.

(r)
Register containing the address value “addr”.

ASCMOD=
Describes the address of a 1-byte field which indicates the ASC (address space control) mode. The addressing mode of the specified process can be read or modified.
The entry in this field has the following meaning:
X'00': program space mode
X'40': access register mode (ASC mode); the program is running in AR mode (see also section “Extended addressing with data spaces” on page 61).

addr
Symbolic address (name) of the field for the ASC mode.

(r)
Register containing the address value “addr”.

PRGCODE=
Defines the address of a field for the program code. The interrupted command or the command set by NIA (**N**ext **I**nstruction **A**ddress) (from the specified process: LAST- or MAIN-PCB) is shown left-justified in this field. Read access only. Field length = 6 bytes. May only be specified in conjunction with the operand PRGCODL.

addr
Symbolic address (name) of the field for the program code.

(r)
Register containing the address value “addr”.

PRGCODL=
Defines the address of a field for the program code. The length of this command is written to this field. Field length = 1 byte.
May only be specified in conjunction with the operand PRGCODE. The following values are possible:

> 0:	(=2 or =4 or =6): Length of the determined program code.
= 0:	Indicator: The program code could not be determined. The field `PRGCODE` remains unchanged.

addr
Symbolic address (name) of the field for the length of the program code.

(r)
Register containing the address value “addr”.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

Layout of the DSECT for LAYOUT=FCONTXT

```
CONTXT LAYOUTF=DSECT
1 *
1 **********************************************************************
1 *           DSECT  FOR  SAVE-FIELD  BY  LAYOUT = FCONTXT              *
1 **********************************************************************
1 *
1 SFCONTXT   DSECT
1 *
1 SAVEHSI    DS    X            HSI INDICATOR
1 SHSI390    EQU   X'01'          /390 HSI
1 SHSI390E   EQU   X'03'          /390 HSI + ESA
1 SHSIRISC   EQU   X'04'          RISC HSI  (NOT USED)
1 SHSISPAC   EQU   X'08'          SPARC HSI
1 SHSISPME   EQU   X'0A'          SPARC HSI + ESA
1 SHSISXI    EQU   X'10'          IA64 HSI
1 SHSISXE    EQU   X'12'          IA64 HSI+ ESA
1 SHSIX86    EQU   X'20'          X86 HSI
1 SHSIX86E   EQU   X'22'          X86 HSI+ ESA
1 *
1 SAVEAMOD   DS    X            ADDRESS MODE
1 SAMODE24   EQU   X'00'          24-BIT ADDRESS MODE
1 SAMODE31   EQU   X'01'          31-BIT ADDRESS MODE
1 *
1 SAVEPMOD   DS    X            PROCESSOR MODE
1 SPMODE1    EQU   X'00'          "/390" (/390 - MACHINE) OR
1 *                               "/390-EMULATION" (SPARC-MACHINE)
1 SPMODE2    EQU   X'01'          "SPARC " (SPARC-MACHINE)
1 *
1 SAVEILC    DS    X            INSTRUCT LENGTH CODE (PCR-FORMAT:BIT 0-1)
1 SAVECC     DS    X            CONDITION CODE (PCR-FORMAT: BIT 2-3)
1 SAVEPM     DS    X            PROGRAM MASK (PCR-FORMAT: BIT 4-7)
1 *
1 SAVEASCM   DS    X            ASC-MODE (ESA)  >> NOT USED ON RISC
1 *
1            DS    XL1          UNUSED
1 *
1 *----------------------------------------------------------------
1 *          PROCESS MODE DEPENDENT AREA
1 *----------------------------------------------------------------
1 *
1 SAV390A    DS    0D         AREA FOR PMODE = /390 AND SPARC
1 *
1 *                           ( RISC )  &  /390   GENERAL REGISTERS
1 *
1 SAVERR0    DS    2F           R0    )       (HARD-WIRED TO ZERO ! )
1 SAVERR1    DS    2F           R1    )
```

```
1 SAVERR2    DS    2F           R2    )
1 SAVERR3    DS    2F           R3    )
1 SAVERR4    DS    2F           R4    >> RISC ONLY
1 SAVERR5    DS    2F           R5    )
1 SAVERR6    DS    2F           R6    )
1 SAVERR7    DS    2F           R7    )
1 *
1 SAVERR8    DS    2F           R8
1 SAVEGR0    EQU   SAVERR8+4      /390:  R0  EQUIVALENT
1 SAVERR9    DS    2F           R9
1 SAVEGR1    EQU   SAVERR9+4      /390:  R1  EQUIVALENT
1 SAVERR10   DS    2F           R10
1 SAVEGR2    EQU   SAVERR10+4     /390:  R2  EQUIVALENT
1 SAVERR11   DS    2F           R11
1 SAVEGR3    EQU   SAVERR11+4     /390:  R3  EQUIVALENT
1 SAVERR12   DS    2F           R12
1 SAVEGR4    EQU   SAVERR12+4     /390:  R4  EQUIVALENT
1 SAVERR13   DS    2F           R13
1 SAVEGR5    EQU   SAVERR13+4     /390:  R5  EQUIVALENT
1 SAVERR14   DS    2F           R14
1 SAVEGR6    EQU   SAVERR14+4     /390:  R6  EQUIVALENT
1 SAVERR15   DS    2F           R15
1 SAVEGR7    EQU   SAVERR15+4     /390:  R7  EQUIVALENT
1 SAVERR16   DS    2F           R16
1 SAVEGR8    EQU   SAVERR16+4     /390:  R8  EQUIVALENT
1 SAVERR17   DS    2F           R17
1 SAVEGR9    EQU   SAVERR17+4     /390:  R9  EQUIVALENT
1 SAVERR18   DS    2F           R18
1 SAVEGR10   EQU   SAVERR18+4     /390:  R10 EQUIVALENT
1 SAVERR19   DS    2F           R19
1 SAVEGR11   EQU   SAVERR19+4     /390:  R11 EQUIVALENT
1 SAVERR20   DS    2F           R20
1 SAVEGR12   EQU   SAVERR20+4     /390:  R12 EQUIVALENT
1 SAVERR21   DS    2F           R21
1 SAVEGR13   EQU   SAVERR21+4     /390:  R13 EQUIVALENT
1 SAVERR22   DS    2F           R22
1 SAVEGR14   EQU   SAVERR22+4     /390:  R14 EQUIVALENT
1 SAVERR23   DS    2F           R23
1 SAVEGR15   EQU   SAVERR23+4     /390:  R15 EQUIVALENT
1 *
1 SAVERR24   DS    2F           R24   )
1 SAVERR25   DS    2F           R25   )
1 SAVERR26   DS    2F           R26   )
1 SAVERR27   DS    2F           R27   >> RISC ONLY
1 SAVERR28   DS    2F           R28   )
1 SAVERR29   DS    2F           R29   )
1 SAVERR30   DS    2F           R30   )       (ADDRESS MODE MASK)
1 SAVERR31   DS    2F           R31   )
```

```
1 *
1 *
1 SAVENIA    DS    2F           NIA/PC (NEXT INSTRUCTION ADDRESS)
1 SVNIA390   EQU   SAVENIA+4      /390:  NIA-EQUIVALENT
1 *
1 *
1 SAVEHI     DS    2F           MULTIPLE/DIVIDE REG HI RESULT (RISC ONLY)
1 SAVELO     DS    2F           MULTIPLE/DIVIDE REG LO RESULT (RISC ONLY)
1 *
1 *
1 *                            ( RISC )  &  /390  FLOATING POINT REGISTERS
1 *
1 SAVEF0     DS    F            F0 : F0/1   = /390: EXT FPR 8  EQUIVALENT
1 SAVEF1     DS    F            F1
1 SAVEF2     DS    F            F2 : F2/3   = /390: EXT FPR 10 EQUIVALENT
1 SAVEF3     DS    F            F3
1 SAVEF4     DS    F            F4 : F4/5   = /390: EXT FPR 12 EQUIVALENT
1 SAVEF5     DS    F            F5
1 SAVEF6     DS    F            F6 : F6/7   = /390: EXT FPR 14 EQUIVALENT
1 SAVEF7     DS    F            F7
1 SAVEF8     DS    F            F8 : F8/9   = /390: EXT FPR 1  EQUIVALENT
1 SAVEF9     DS    F            F9
1 SAVEF10    DS    F            F10: F10/11 = /390: EXT FPR 3  EQUIVALENT
1 SAVEF11    DS    F            F11
1 SAVEF12    DS    F            F12: F12/13 = /390: EXT FPR 5  EQUIVALENT
1 SAVEF13    DS    F            F13
1 SAVEF14    DS    F            F14: F14/15 = /390: EXT FPR 7  EQUIVALENT
1 SAVEF15    DS    F            F15
1 SAVEF16    DS    F            F16: F16/17 = /390: EXT FPR 9  EQUIVALENT
1 SAVEF17    DS    F            F17
1 SAVEF18    DS    F            F18: F18/19 = /390: EXT FPR 11 EQUIVALENT
1 SAVEF19    DS    F            F19
1 *
1 SAVEF20    DS    2F           F20/21  =   /390:  FPR 0  EQUIVALENT
1 SAVEF22    DS    2F           F22/23  =   /390:  FPR 2  EQUIVALENT
1 SAVEF24    DS    2F           F24/25  =   /390:  FPR 4  EQUIVALENT
1 SAVEF26    DS    2F           F26/27  =   /390:  FPR 6  EQUIVALENT
1 *
1 SAVEF28    DS    F            F28: F28/29 = /390: EXT FPR 13 EQUIVALENT
1 SAVEF29    DS    F            F29
1 SAVEF30    DS    F            F30: F30/31 = /390: EXT FPR 15 EQUIVALENT
1 SAVEF31    DS    F            F31
1 *
1 SAVEFCR    DS    F          FP-CONTROL-/STATUS REG >> NOT USED ON RISC
1 *
1 *
1 *                            /390-ESA  ACCESS REGISTERS
1 *
```

```
1 SAVEAR0    DS    F            ACR0       )
1 SAVEAR1    DS    F            ACR1       )
1 SAVEAR2    DS    F            ACR2       )
1 SAVEAR3    DS    F            ACR3       >> NOT USED ON RISC
1            DS    9F           ACR4-ACR12 )
1 SAVEAR13   DS    F            ACR13      )
1 SAVEAR14   DS    F            ACR14      )
1 SAVEAR15   DS    F            ACR15      )
1 *
1 SWOSPARC   EQU   *-SFCONTXT   LENGTH   -  WITHOUT SPARC-BLOCK
1 *
1 *-----------------------------------------------------------------
1 *          SPARC  CONTEXT  BLOCK
1 *-----------------------------------------------------------------
1 *
1            DS    0D                                  )
1 SSPARCB    DS    100D         SPARC-AREA: BEGIN  )
1            DS    100D                                >> FOR SPARC ONLY
1            DS    15D                                 )
1 SSPARCE    DS    0D           SPARC-AREA: END    )
1 SLSPARCB   EQU   SSPARCE-SSPARCB       LENGTH OF SPARC-BLOCK (SPARC)
1 *
1 *-----------------------------------------------------------------
1 *
1 SAVLNGTH   EQU   *-SFCONTXT   LENGTH  OF  SAVE-FIELD (SPARC)
1 *
1 *-----------------------------------------------------------------
1 *         IA64 CONTEXT  BLOCK: FCONTEXT
1 *-----------------------------------------------------------------
1 *
1            ORG   SSPARCB      REDEFINITION OF NATIVE AREA
1 SSXIB      DS    0D         IA64-AREA: BEGIN
1 SLIA64     EQU   SWOSPARC+4064        LENGTH OF IA64 CONTEXT
1 *
1 *
1 *
1 *-----------------------------------------------------------------
1 *         X86  CONTEXT  BLOCK: FCONTEXT
1 *-----------------------------------------------------------------
1 *
1            ORG   SSPARCB      REDEFINITION OF NATIVE AREA
1 SX86E      DS    0D         X86-AREA: BEGIN
1 SLX86E     EQU   SWOSPARC+4096        LENGTH OF X86 CONTEXT
1 *
1 *-----------------------------------------------------------------
```

Return information and error flags

During execution of the macro, register R1 contains the address of the operand list.

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the CONTXT macro is transferred in register R15.
aa=X'00': normal execution
aa≠X'00': function was not executed.

X'bb'	X'aa'	Meaning
X'00'	X'00'	Normal execution. The PCB had not yet been changed (in the current interrupt state)
X'04'	X'00'	Normal execution. The PCB had already been changed with CONTXT (in the current interrupt state)
X'04'	X'04'	Function was not executed; invalid operands
X'04'	X'08'	Function was not executed. The macro was given in a basic process
X'04'	X'18'	Write not permitted This may have the following causes: – writing of BD, PRGCODE – writing of CC, ILC, PM in x86 PCB – writing of a program counter which points to a x86 module when LAYOUT=COMPATIBLE is specified
X'04'	X'1C'	Function was not executed. When FUNCT=WRITE the storage key in the PCB to be changed does not match the storage key in the current PCB
X'04'	X'20'	Function was not executed: it is not possible to access the access registers

PCB accesses in the /390 process mode

The following table shows the result of accessing the PCB, depending on the CONTXT operands, run mode of the interrupted process and the LAYOUT operand specification:

CONTXT operand	CONTXT FUNCT=	LAYOUT=COMPATIBLE		LAYOUT=FCONTXT	
		/390-PCB	x86 PCB	/390 PCB	x86 PCB
SAVE	READ	(1)	(2)	(3)	(4)
	WRITE		RC=X'18' (*)		
OWNR/STACKR (without PC/NIA)	READ	as before	R0-R15	as before	R0-R15
	WRITE				
STACKR=(PC)	READ	as before	x86 native NIA	as before	x86 nativeNIA
	WRITE		RC=X'18' (*)		
SAVACR/ OWNACR /	READ	as before	as before	as before	as before
STKACR / ASCMOD	WRITE				
FPR	READ	(5)	(5)	(6)	(7)
	WRITE				
ILC	READ	as before	value: 0 or 4	as before	value: 0 or 4
	WRITE		RC=X'18'		RC=X'18'
CC	READ	as before	value: X'00'	as before	value: X'00'
	WRITE		RC=X'18'		RC=X'18'
PM	READ	as before	value: X'00'	as before	value: X'00'
	WRITE		RC=X'18'		RC=X'18'
MODE	READ	as before	as before	as before	as before
	WRITE				
PMODE	READ	X'00'	X'01'	X'00'	X'01'
	WRITE	(8)	(8)	(8)	(8)

Meaning of the abbreviations used in the table:

R0 - R15:	general registers
RC:	Return code

Meaning of the comments in the table:

(*): The program counter (PC / NIA = Next Instruction Address) may only be written if it points to a /390 module (with `LAYOUT= COMPATIBLE`) or if `LAYOUT=FCONTXT` has been specified.

(1): The data exchange field has the same structure as before.

(2): The data exchange field has the same structure as before.
Only those registers that are equivalent to /390 registers and NIA are read or written.

(3): The data exchange field utilizes the new structure. Only the registers equivalent to the /390 registers, the x86 floating-point registers relevant to the /390 floating-point registers, NIA and the other process statuses are read or written (provided they can be overwritten). Data is exchanged via the corresponding `SAVExxxx` fields.

(4): In addition to (3), the complete x86 context is read and written (via the exchange area `SSPARCB` of the length `SLSPARCB`). Modifying individual data in the context is only possible via the `SAVExxxx` fields.

(5): The data is transferred between the /390 floating-point registers of the specified PCB and those of the contingency PCB.

(6): Only via the SAVE operands, see (3)

(7): Only via the SAVE operands, see (4)

(8): Writing the PMODE is allowed in order, for example, to restart the interrupted PCB at a central termination routine, whose PMODE is not the interrupted PMODE. Writing the PMODE only makes sense if the NIA is modified at the same time.

CRYPT – Word encryption

General

Application area: Word encryption; see page 161
Macro type: Type S, MF format **3**: D/C/M/E/L form; see page 29

Macro description

The **CRYPT** macro is used for one-way encryption of words with a maximum length of 8 bytes. One-way encryption means that it is not possible to decrypt the words that have been encrypted with **CRYPT**. A 4-byte or 8-byte string is returned as a result of macro execution.

Macro format and description of operands

```
CRYPT

INSTRA=<var: pointer> / (<reg: pointer>)
,INSTRL=4 / 8 / <var: int:1>
,OUSTRA=<var: pointer> / (<reg: pointer>)
,CRYALG=*SCA / *SCAVK / *OLD / <var: enum-of _ecrt_s:1>
,CRCL2OP=*YES / *NO / <var: enum-of _cl2op_s:1>
,VKEYA=<var: pointer> / (<reg: pointer>)
,XPAND=*INPAR / *KEYPAR
,MF=D / C / M / E / L
[,PARAM = addr / (r)]
,PREFIX=S / p
,MACID=RME / macid
```

The operands are described in alphabetical order below.

CRCL2OP=
Encrypts the input word in accordance with the system parameter ENCRYPTION (see the “Introduction to System Administration” manual [10]).

***YES**
Encryption is in accordance with the system parameter ENCRYPTION.

***NO**
The input word is always encrypted, regardless of the system parameter ENCRYPTION.

<var: enum-of _cl2op_s:1>
Name of the field together with the type of encryption.

CRYALG=
Selects the encryption algorithm.

***SCA**
The encryption algorithm SCA is used.

***SCAVK**
The encryption algorithm SCA is used with the key specified in the key field (VKEYA operand).

***OLD**
Specifies the encryption algorithm used previously.

<var: enum-of _ecrt_s:1>
Name of the field with the encryption algorithm.

INSTRA=
Defines the address of a field which contains the word which is to be encrypted (input word). The length of this field is specified in the INSTRL operand. The input word must be a string of type X string or C string and may be a maximum of 8 characters in length.
This operand is mandatory in conjunction with MF=L.

<var: pointer>
Name of the field with the address of the input word; only permitted with MF=M.

(<reg: pointer>)
Register with the address of the input word; only permitted with MF=M.

INSTRL=
Specifies the length of the field that has to be reserved for the encrypted input word. the maximum permitted value for the length is 8. The field in which the input word is now specified may now be either 4 or 8 bytes long

4
4 bytes are reserved for the field length.

8
8 bytes are reserved for the field.

<var: int:1>
Name of the field together with a specification of the field length that has to be reserved.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form or D form of the macro and additionally a MACID in the C form (see section “S-type macros” on page 29).

OUSTRA=
Defines the address of a field which is to contain the encrypted word (output word). If the input word is ≤ 4 bytes, a 4-byte long output word is returned. If the length of the input word is between 5 and 8 bytes, an 8-byte long output word is returned.
These operands are mandatory in conjunction with MF=L.

<var: pointer>
Name of the field with the address of the output word; only permitted with MF=M.

(<reg: pointer>)
Register with the address of the output word; only permitted with MF=M.

VKEYA=
Address of the employed variable key which is to be used if the encryption setting CRYALG=*SCAVK is selected.

<var: pointer>
Name of the field with the address of the variable key; only permitted with MF=M.

(<reg: pointer>)
Register with the address of the variable key; only permitted with MF=M.

XPAND=
Controls the expansion scope of the macro.

***INPAR**
Preset value: the parameter structure is expanded.

***KEYPAR**
Only the data area for the variable key is expanded.

Selecting the variable key

A variable key may be used only in conjunction with the SCA encryption algorithm (CRYALG=*SCA operand). The variable key is 44 bytes long and comprises the following four parts (PREFIX and MACID are assigned their respective default settings):

```
SRMEVK      DS    0F
SRMECC      DS    F          number of iterations
SRMEKEE1    DS    XL16       key component EE1
SRMEKEE2    DS    XL16       key component EE2
SRMEKEE3    DS    XL8        key component EE3
SRMEVK#     EQU   *-SRMEVK
```

The SCA encryption algorithm is an iterative application of a basic encryption method. To provide sufficient security, the number of iterations (SRMECC field) must be between 128 and 8192. Encryption using 128 iterations requires approximately 15000 operations. The number of operations increases linearly with the number of iterations.

The key components EE1 and EE2 (SRMEKEE1 and SRMEKEE2 fields) represent permutations of the numbers 0 through 15. For a “secure” key, each byte of EE1 and EE2 must contain a number between 0 and 15; each of these numbers must occur once in each of the key components EE1 and EE2.
The key component EE3 (SRMEKEE3 field) can contain any characters and has a length of 8 bytes. However, no two of these bytes may be identical.

The encryption routine does not check whether the above conditions for a “secure” variable key have been met. It is often impossible to avoid using “insecure” keys in one-way encryption. For this reason, the SCA encryption algorithm can also work with “insecure” keys. For reasons of security, however, the use of such keys should be avoided if possible.

Encryption of input words > 8 bytes long

If encryption of words > 8 bytes long is required, it is possible to split the input word into several 8-byte words and encrypt these word segments separately. The encryption of words > 8 bytes long is no more secure than the encryption of words ≤ 8 bytes long using the algorithms provided by the **CRYPT** macro.

Layout of the DSECT

The layout of the DSECT is to be found on page 345.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A structured return code relating to the execution of the CRYPT macro is returned:
(cc=Subcode2, bb=Subcode1, aaaa=Maincode)

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function executed successfully
X'01'	X'01'	X'0001'	Function not executed due to operand error: input word not assigned
X'02'	X'01'	X'0001'	Function not executed due to operand error: output word not assigned
X'03'	X'01'	X'0001'	Function not executed due to operand error: no assignment for variable key
X'04'	X'01'	X'0001'	Function not executed due to operand error: invalid specification for encryption algorithm
X'05'	X'01'	X'0001'	Function not executed due to operand error: invalid specification for dependency on system parameter
X'06'	X'01'	X'0001'	Function not executed due to operand error: invalid length specification for input word
X'08'	X'01'	X'0001'	Function not executed due to operand error: invalid variable key
X'09'	X'01'	X'0002'	Function not executed due to memory request error: it is not possible to access the field containing the input word
X'0A'	X'01'	X'0002'	Function not executed due to memory request error: it is not possible to access the field containing the output word
X'0B'	X'01'	X'0002'	Function not executed due to memory request error: it is not possible to access the field containing the variable key
X'0C'	X'01'	X'0002'	Function not executed due to memory request error: it is not possible to access the parameter list
X'0D'	X'01'	X'0002'	Function not executed due to memory request error it is not possible to access the SRMEPOE field)
	X'20'	X'0003'	Function not executed: internal error

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Example

```
PRINT NOGEN
   CRYPT    START
            BALR  10,0
            USING *,10
            CRYPT MF=E,PARAM=PARLIST ----------------------------------   (1)
 1          MFCHK MF=E,PREFIX=S,MACID=RME,PARAM=PARLIST,
 1                SVC=16,
 1                DMACID=RME,SUPPORT=(D,L,C,M,E)
 2          LA    1,PARLIST
 2          SVC   16
            CLI   SRMEMR1,SRMEOK      * Error query
            BNE   ERREXIT
            UNPK  OUTPUTX(9),OUTPUT(5)
            UNPK  OUTPUTX+8(9),OUTPUT+4(5)
            TR    OUTPUTX,CODETAB-C'0'
            WROUT CODE,0              * Output
            TERM
   *
   ERREXIT  WROUT TEXT,0
            TERM
   ****
   CODE     DC    Y(CODEEND-CODE)
            DS    CL3
            DC    C'OUTPUT OF THE ENCRYPTED WORD '
   INPUT    DC    C'SUPERMAN'         * Input word *
            DC    C': '
   OUTPUT   DS    CL8                 * Output word
            DC    C'  '
   OUTPUTX  DS    CL16                * Output word hex
   CODEEND  EQU   *
            DS    C
   TEXT     DC    Y(TEXTEND-TEXT)
            DS    CL3
            DC    C'ERROR !!'
   TEXTEND  EQU   *
   KEY      DS    0F
            DC    F'250'              * Number of iterations
            DC    X'0203040506070809' * EE1
            DC    X'0A0B0C0D0E0F0001'   EE1 *
            DC    X'0100030205040706' * EE2
            DC    X'09080B0A0D0C0F0E'   EE2 *
            DC    X'A1A2A3A4A5A6A7A8' * EE3 *
   PARLIST  CRYPT MF=L,INSTRL=8,CRYALG=*SCAVK,CRCL2OP=*NO, -
                  VKEYA=KEY,INSTRA=INPUT,OUSTRA=OUTPUT ----------------   (1)
            ORG   PARLIST
```

```
           CRYPT MF=C  ────────────────────────────────────────────────  (2)
1 *
1 SRMEPA     DS    OF          BEGIN of PARAMETERAREA
1         FHDR  MF=(C,SRME),EQUATES=NO                    STANDARD HEADER
2          DS    0A
2 SRMEFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 SRMEIFID DS    0A              0   INTERFACE IDENTIFIER
2 SRMEFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 SRMEFCT  DS    AL1             2   FUNCTION NUMBER
2 SRMEFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 SRMERET  DS    0A              4   GENERAL RETURN CODE
2 SRMESRET DS    0AL2            4   SUB RETURN CODE
2 SRMESR2  DS    AL1             4   SUB RETURN CODE 2
2 SRMESR1  DS    AL1             5   SUB RETURN CODE 1
2 SRMEMRET DS    0AL2            6   MAIN RETURN CODE
2 SRMEMR2  DS    AL1             6   MAIN RETURN CODE 2
2 SRMEMR1  DS    AL1             7   MAIN RETURN CODE 1
2 SRMEFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1 *       RETURN CODE EQUATES FOR MAIN-CODE 1
1 SRMEOK  EQU   X'00'               NOERROR
1 SRMEIOP EQU   X'01'               INVALID OPERAND
1 SRMEIAR EQU   X'02'               INVALID AREA
1 SRMEINE EQU   X'03'               INTERNAL ERROR
1 *
1 SRMEIN  DS    F                   INPUT STRING ADDRESS
1 SRMEOUT DS    F                   OUTPUT STRING ADDRESS
1 SRMELEN DS    X                   INPUT STRING LENGTH
1 SRMEECR DS    X                   SELECT ENCRYPTION ROUTINE
1 *
1 *       EQUATES FOR ENCRYPTION ROUTINE SELECT
1 SRMEECRS EQU   X'01'               SCA ENCRYPTION ROUTINE
1 SRMEECRO EQU   X'02'               OLD ENCRYPTION ROUTINE
1 SRMEECRV EQU   X'03'               SCA ENCR. ROUT. (V. KEY)
1 *
1 SRMEC2O DS    X                   SEL. CLASS 2 OPTION YES/NO
1 *       EQUATES FOR CLASS 2 OPTION
1 SRMEC2OY EQU   X'01'               CLASS 2 OPTION YES
1 SRMEC2ON EQU   X'02'               CLASS 2 OPTION NO
1 *
1 SRMEPOE DS    X                   PROGRESS OF EXECUTION
1 *       RETURN FOR PROGRESS OF EXECUTION
```

```
1 SRMEUV  EQU   X'01'               UNCRYPTED
1 SRMESCA EQU   X'02'               SCA ENCRYPTED
1 SRMEOLD EQU   X'03'               OLD ENCRYPTED
1 SRMESVK EQU   X'04'               SCA ENCRYPTED (V. KEY)
1 *
1 SRMESVK@ DS    F                   ADR. VARIABLE KEY
1 SRME#   EQU   *-SRMEPA    LENGTH OF PARAMETERAREA
  *
  CODETAB  DC    C'0123456789ABCDEF'
           END   CRYPT
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
%//compile source=*library-element(lib.srpmencp,crypt), -
%//        compiler-action=module-generation(module-format=llm), -
%//        module-library=lib.srpmencp, -
%//        listing=parameters(output=*library-element(lib.srpmencp,crypt))
%  ASS6011 ASSEMBLY TIME: 360 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 65 MSEC
%//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=lib.srpmencp,element-or-symbol=crypt
%  BLS0523 ELEMENT 'CRYPT', VERSION '@', TYPE 'L' FROM LIBRARY
   ':20SC:$EVA.LIB .SRPMENCP' IN PROCESS
%  BLS0524 LLM 'CRYPT', VERSION ' ' OF '<date> <time>' LOADED
OUTPUT OF THE ENCRYPTED WORD SUPERMAN: |ÑzÄkè~]  4F69A9639254FFBD ─────  (3)
```

(1) A word with a length of 8 bytes is to be encrypted with the **CRYPT** macro using the SCA algorithm, irrespective of the class 2 option. A variable key is to be used. The input word is read from the `INPUT` field and the output word is written to the `OUTPUT` field.

(2) Layout of the DSECT.

(3) The `CODE` field is output by means of the **WROUT** macro. The input word `SUPERMAN` is encrypted as |ÑzÄkè~]. The hexadecimal result of the encryption is `4F69A9639254FFBD`.

CSTAT – Change page status

General

Application area:	Working with virtual memory; see page 55 Memory pools; see page 55
Macro type:	Type S, MF format **1**: standard/E/L form; see page 29

The operating system manages virtual storage on a page basis (among other things, this involves paging, memory protection, requests for pages in memory and their release). One page of memory is 4 K (=4096 bytes). Space is assigned to the user program in class 6 memory in page-sized portions.

Macro description

The **CSTAT** macro enables users to amend the attributes of the pages assigned to their program in class 6 memory in relation to

- paging management (whether the pages are resident or pageable)
- mode of access (read/write access)
- special access protection (accesses via AID or DUMP only allowed with special privilege).

For memory pages which are in a memory pool, the following points should be noted:

- Memory pages in a resident memory pool can be made nonpageable by a **CSTAT** call (see also the macro **ENAMP**). When the macro is executed, only those pages which lie outside the memory pool are affected.
- Memory pages in a nonresident memory pool can be made resident and subsequently pageable again by any memory pool user issuing **CSTAT**.
- Where changes to access rights are concerned, the **CSTAT** macro has a lower priority than the **CSTMP** macro:
  A write protection set up for a memory pool using **CSTMP** cannot be removed (page by page) using **CSTAT**; this even applies to pages whose write protection was set up using **CSTAT** before the call to **CSTMP**.

Macro format and description of operands

CSTAT
PGNUM=value / ALL / (r) [,ACCESS=READ / WRITE / (r)] [,PAGE=YES / NO / (r)] [,PROTECT=YES] [,MF=(E,..) / L]

PGNUM=
Specifies the page number(s) of the page(s) whose status is to be amended in respect of paging, mode of access or special access protection. In addition to PGNUM, at least one further operand must be specified. A call to **CSTAT** containing only PGNUM=.... is meaningless, and will result in the error flag (return code value) X'0C'.

value
Page number of the page whose status is to be amended.

ALL
The status of all the pages used by the program is to be amended. The operand PGNUM=ALL must only be used in combination with PAGE=...

(r)
Register which contains the page number (value) or the parameter ALL (X'40C1D3D3'); (2 ≤ r ≤ 12).

ACCESS=
Determines whether the specified pages may be accessed only for reading, or may also be written to (write access implies permission to read).

Notes

- ACCESS=... must not be specified in combination with PGNUM=ALL.
- Memory pool: a write protection established by **CSTMP** cannot be removed using **CSTAT**.

READ
Only read access is allowed.

WRITE
Write access is allowed.

(r)
Register which contains the parameter YES (X'40E8C5E2) or NO (X'4040D5D6').
YES: write access allowed.
NO: only read access allowed.

PAGE=

Specifies whether the memory pages are to be pageable or resident.

YES

The memory page(s) is (are) to be pageable.
Memory pages which are part of a resident memory pool are unaffected.

NO

The memory page(s) is (are) to be resident.
The maximum number of pages that can be made resident is limited by the operand RESIDENT-PAGES=PARAMETERS(MINIMUM=...) in the command START-PROGRAM or LOAD-PROGRAM.

Note

If memory saturation should occur, the following steps can be taken:

- Any pending **CSTAT** requests for resident memory pages are rejected (in spite of this, the value of the return code will be X'00').
- Some of those memory pages of a program which have already been made resident with **CSTAT** may be made pageable by the operating system (their resident status will not automatically be restored later).

(r)

Register containing the parameter YES (X'40E8C5E2') or NO (X'4040D5D6');
(2 ≤ r ≤ 12).

PROTECT=

Specifies a special access protection for the memory page.

Notes

- PROTECT=YES must not be used in combination with PGNUM=ALL or ACCESS=....
- Any access protection set up using PROTECT can only be removed by releasing the memory page (**RELM**). This will cause the contents of the page to be deleted!

YES

Accesses to the memory page by the debugging aid AID are only permitted if special privilege is assigned. The same is true for a memory dump.

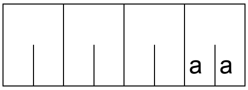
MF=

For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

Return information and error flags

During execution, the register R1 contains the address of the operand list.

R15:



A return code relating to the execution of the CSTAT macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Normal execution
X'04'	The specified page does not belong to the program, and was also not requested using REQM
X'0C'	Operand error
X'10'	The program has attempted to make more pages resident then were reserved for it in the START-PROGRAM/LOAD-PROGRAM command (operand RESIDENT-PAGES)

CSTMP – Set read/write access for memory pool

General

Application area:	Memory pools; see page 55
Macro type:	Type S, MF format **1**: standard/L/D/E form; see page 29

A memory pool (MP) is a memory area in class 6 memory that can be used by several users at the same time. Its size (and position) is determined by the first user. A memory pool can be furnished with write protection (**CSTMP** has priority over **CSTAT**).

Creating a “read-only memory pool”:

1. create the MP and request memory pages (**ENAMP**, **REQMP**)
2. write to the MP (e.g. load shared code)
3. establish write protection = set MP to “read only” (**CSTMP**)

Macro description

The (authorized) user can provide a memory pool with write protection (only read access is allowed) or revoke this protection with the **CSTMP** macro. The requested access protection applies to all pages of the memory pool and all participants. The function is only carried out if the user has the required authorization (CSTMP=YES) in the user catalog.

Notes

- A memory pool is addressed by means of either its pool name or its ID (see **ENAMP**).
- **CSTMP** overrides the effect of the **CSTAT** macro:
  - **CSTAT** is rejected if write protection has already been established with **CSTMP**.
  - write protection established with **CSTAT** can be removed or extended to all pool pages by means of **CSTMP**. Such an extended write protection can only be removed again by means of **CSTMP**.
- Different levels of protection cannot be established with **CSTMP**.
- It is not possible to either request (**REQMP**) or release (**RELMP**) pages for a write-protected memory pool.

Macro format and description of operands

```
CSTMP
------------------------------------------------------------------------------------
 ⎧ ⎧ MPNAME=name                         ⎫                                                ⎫
 ⎨ ⎩ MPNAMAD=addr [,MPNAMLN=length]      ⎭,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL      ⎬
 ⎩ MPID=addr                                                                              ⎭

,ACCESS=WRITE / READ
[,PARMOD=24 / 31]
[,MF=L / (E,..) / (D,pre) / D]
```

MPNAME=
Defines the name of the memory pool.

name
Name of the memory pool (note the connection with the SCOPE operand).

MPNAMAD=
Specifies the address of the field containing the name of the memory pool.

addr
Symbolic address (name) of the field (note the connection with the SCOPE operand).

MPNAMLN=
Gives the length of the name to which MPNAMAD refers. If this operand is omitted, the length attribute of “addr” is assumed.

length
Length in bytes.

MPID=
Defines the address of the field (length = 4 bytes) with the ID for the memory pool (see also **ENAMP**). The ID identifies the memory pool uniquely. The use of the memory pool ID increases the speed of processing.

addr
Symbolic address (name) of the field containing the ID.

SCOPE=
Defines the scope (authorized users) of the memory pool. This specification is used to identify the memory pool uniquely and must always be entered in conjunction with the MPNAME or MPNAMAD operand.

LOCAL
The memory pool is used only by the user that created it.

GROUP
Memory pool users can be all tasks with the ID of the user that created the memory pool.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.
The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

ACCESS=
Defines whether the memory pool is read-only or open to both read and write access (authorization for write access implies read access as well); this applies to all users.

WRITE
Write access is permissible.

READ
Only read access is permissible.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A prefix (pre = 1..3 letters) can be specified in the D form of the macro, as shown in the macro format.
Default setting: pre = CST

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Return information and error flags

After macro processing, register R1 contains the operand list address.

R15:

b	b	0	0	0	0	a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the CSTMP macro is transferred in register R15.
aa=X'00': normal execution;
aa=X'04': function is not executed.

X'bb'	X'aa'	Meaning
X'00'	X'00'	Normal execution
X'04'	X'04'	Function is not carried out; the caller is not a memory pool user (no ENAMP macro)
X'1C'	X'04'	Function is not carried out; operand error: – invalid address of the operand list – error in operand list structure – invalid address for MPNAMAD or MPID in the operand list – designation of the memory pool: – name contains invalid characters – invalid length specification (MPNAMLN) – memory pool not designated (MPNAME, MPNAMAD, MPID not specified) – MPNAMLN specified, but MPNAMAD omitted – SCOPE specified, but MPNAME/MPNAMAD omitted – designation not unequivocal: more than one operand was specified as designation (MPNAME/MPNAMAD/MPID) – invalid SCOPE specification – SCOPE=USER_GROUP was specified, but SRPM is not available in system – invalid ACCESS specification – PARMOD=24 specified in conjunction with 31-bit addressing mode (AMODE31)
X'24'	X'04'	Function not carried out; authorization error: – the caller is not authorized to call the CSTMP macro (missing entry in the user ID) – the caller is not authorized to change the access protection of a privileged or class 5 memory pool – the access key at the time of the CSTMP call does not match the access key which was valid when the memory pool was established

31-bit interface:

- In the event of errors in the alignment or initialization of the standard header, the return codes X'0001FFFF' / X'0003FFFF'/ X'0004FFFF' are additionally transferred in register R15; see table “Standard return codes” on page 43.
- No return codes are transferred in the standard header.

CTIME – Time stamp calculations

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Type S, MF format **3**: C/D/E/L/M form; see page 29

Macro description

The **CTIME** macro offers the following functions:

- conversion of the time stamp format (FUNCT=*CONV)
- addition and subtraction of a timespan to/from a time stamp (FUNCT=*ADD)
- calculation of the timespan between two time stamps (FUNCT=*DIFF)

CTIME supplies the results in the operand list in printable, binary or TODR/TODX form.

Notes

- The **CTIME** macro does not use an SVC.
  If MF=D (default setting), the system does not initialize a standard header, i.e. the user must define the standard header before calling the macro (for information on the structure of the standard header, see page 43).
- Because the functions of the **CTIME** macro are initiated via a subprogram interface rather than a SVC, a program calling **CTIME** must provide a save area 18 words long. The address of this save area must be loaded into register R13 before the macro call.

The operand list contains memory areas for:

Input time stamp 1	`NTICS1I`	Length:	48 bytes
Input time stamp 2	`NTICS2I`		48 bytes
Input timespan	`NTICD1I`		32 bytes
Output time stamp	`NTICS1O`		48 bytes
Output timespan	`NTICD1O`		32 bytes

The field names for PREFIX=N and MACID=TIC apply.

Time stamps and timespans are used to define the DSECTs NTICS (for time stamps) and NTICD (for timespans) for addressing data fields containing time specifications.

A time stamp consists of the date (year, month, day, day of the week) and time (hours, minutes, seconds, milliseconds, microseconds). It always refers to a particular time base and can be specified in different formats. Timespans consist of day and time specifications and can also be specified in different formats.

Possible **time bases** are:

UTC: Universal Time Coordinate ≙ Greenwich Mean Time (world time)
LTI: Local Time (local time in the calling system)
FZ: Foreign Zone (“foreign” or any time base)
If the FZ time base is specified, zone information must also be specified to make the time stamp unambiguous. This information indicates which world time zone difference and/or which season difference must be observed and whether the time stamp was specified in summer or winter time.

Time stamps can have the following formats:

ISO4 Printable format in decimal notation as follows:
yyyy-mm-ddjjj wwhh:mm:ssvhz:mz-hs:ms-smlsmcs

where

`yyyy`	Year	(4 bytes)
`mm`	Month	(2 bytes)
`dd`	Day of the month	(2 bytes)
`jjj`	Julian date	(3 bytes)
`ww`	Day of the week	(2 bytes)
`hh`	Hours	(2 bytes)
`mm`	Minutes	(2 bytes)
`ss`	Seconds	(2 bytes)
`v`	Sign of zone difference	(1 bytes)
`hz`	Hours of zone difference	(2 bytes)
`mz`	Minutes of zone difference	(2 bytes)
`hs`	Hours of season difference	(2 bytes)
`ms`	Minutes of season difference	(2 bytes)
`s`	Summer/wintertime specification	(1 bytes)
`mls`	Milliseconds	(3 bytes)
`mcs`	Microseconds	(3 bytes)

The data area for accepting the time stamp has the following format (macro expansion with MF=D and PREFIX=N):

```
NTICSISO4       DS    0XL48         timestamp in iso4
NTICSDATE_U     DS    0XL10         date union
*
NTICSMD         DS    0XL10         date
NTICSIDY        DS    CL4           year
NTICSID1        DS    CL1           hyphen1
NTICSIDM        DS    CL2           month
NTICSID2        DS    CL1           hyphen2
NTICSIDD        DS    CL2           day
*
                ORG   NTICSDATE_U
NTICSDATE_CHAR  DS    CL10          date_char
                ORG   NTICSDATE_U+10
NTICSIDJ        DS    CL3           julian
NTICSIDB        DS    CL1           blank
*
NTICSMW         DS    0CL2          begin of weekday
NTICSIWD        DS    CL2           weekday
*
NTICSMT         DS    0CL8          time: "hh:mm:ss"
NTICSITH        DS    CL2           hour
NTICSIT1        DS    CL1           colon1
NTICSITM        DS    CL2           minute
NTICSIT2        DS    CL1           colon2
NTICSITS        DS    CL2           second
*
NTICSMZ         DS    0CL14         zone: "shh:mm-hh:mm-a"
NTICSIZS        DS    CL1           zonesign
NTICSIZH        DS    CL2           zonehour
NTICSIZ1        DS    CL1           colon3
NTICSIZM        DS    CL2           zoneminute
NTICSIZ2        DS    CL1           hyphen3
NTICSISH        DS    CL2           seasonhour
NTICSIS1        DS    CL1           colon4
NTICSISM        DS    CL2           seasonminute
NTICSIS2        DS    CL1           hyphen4
NTICSISA        DS    FL1           actualseason
*
NTICSMF         DS    0XL6          begin of fraction of second
NTICSIF         DS    0CL6          fraction of second : "mmmuuu"
NTICSIFM        DS    CL3           millisecond
NTICSIFN        DS    CL3           microsecond
NTICSILE        EQU   *-NTICSISO4   LENGTH OF ISO4 TIMESTAMP
*                                   (without address of CHDATE List)
NTICSCDL        DS    A             chdates_addr
```

BINAR The numeric values are specified in halfwords in binary notation.

The data area for accepting the time stamp has the following format (macro expansion with MF=D and PREFIX=N):

```
NTICSBINAR      DS    0XL42        timestamp in binar
NTICSBDY        DS    H            year
NTICSBDM        DS    H            month
NTICSBDD        DS    H            day
NTICSBDJ        DS    H            julian
NTICSFILL1      DS    CL6          fill1
NTICSBWD        DS    H            weekday
*
NTICSBTH        DS    H            hour
NTICSBTM        DS    H            minute
NTICSBTS        DS    H            second
NTICSBL1        EQU   *-NTICSBINAR LENGTH OF BINARY TIMESTAMP
*                                     PART1 (date & time)
NTICSFILL2      DS    CL2          fill2
NTICSBZH        DS    H            zonehour
NTICSBZM        DS    H            zoneminute
NTICSBSH        DS    H            seasonhour
NTICSBSM        DS    H            seasonminute
NTICSBSA        DS    FL1          actualseason
NTICSFILL3      DS    CL5          fill3
*                                  fraction of second
NTICSBFM        DS    H            millisecond
NTICSBFN        DS    H            microsecond
NTICSBL2        EQU   *-NTICSBFM   LENGTH OF BINARY TIMESTAMP
*                                     PART2 (fraction of second)
NTICSBLE        EQU   *-NTICSBINAR LENGTH OF BINARY TIMESTAMP
```

TODR Time of day register format. The TOD register has the length of a doubleword and contains:
`((Number of microseconds since 1/1/1900) * 4096) modulo 2`^{64}

It is incremented by the hardware. The TOD register can be queried using the STCK command. The interpretation of the TODR content depends on the epoch set for the system run, see the “Introduction to System Administration” manual [10]).

The data area for accepting the time stamp has the following format (macro expansion with MF=D and PREFIX=N):

```
NTICST          DS    0XL8         timestamp in TODR
NTICSTMS        DS    F            most significant word:
*                                     approx sec
NTICSTLS        DS    F            least significant word:
*                                     micro_sec * 2**12
```

TODX Extended time of day register format: it has the length of a doubleword and contains the number of microseconds since January 1 1900 0 hours.

The data area for accepting the time stamp has the following format (macro expansion with MF=D and PREFIX=N):

```
NTICSX          DS    0XL8         timestamp in TODX
NTICSTHW        DS    F            high word
NTICSTLW        DS    F            low word
```

Timespans can have the following formats:

ISO4 / ISO4MIC Printable format in decimal notation.
The data area for accepting the timespan has the following format (macro expansion with MF=D and PREFIX=N):

```
NTICDI        DS    0XL32        tdiff in iso4
NTICDIS       DS    CL1          sign
NTICDIDD      DS    CL10         day
NTICDID1      DS    CL1          hyphen
NTICDITH      DS    CL2          hour
NTICDIT1      DS    CL1          colon1
NTICDITM      DS    CL2          minute
NTICDIT2      DS    CL1          colon2
NTICDITS      DS    CL2          second
NTICDIFP      DS    CL1          point
NTICDIFM      DS    CL3          millisecond
NTICDIFN      DS    CL3          microsecond
NTICDILE      EQU   *-NTICDI     LENGTH OF ISO4 TIMEDIFFERENCE
*                                (without unused field)
              DS    CL5          unused
```

BINAR / BINARMIC Binary specification.
The data area for accepting the timespan has the following format (macro expansion with MF=D and PREFIX=N):

```
NTICDB        DS    0XL16        tdiff in binar
NTICDBDD      DS    F            day
NTICDBTH      DS    H            hour
NTICDBTM      DS    H            minute
NTICDBTS      DS    H            second
NTICDBFM      DS    H            millisecond
NTICDBFN      DS    H            microsecond
NTICDBLE      EQU   *-NTICDB     LENGTH OF BINARY TIMESTAMP
*                                (without the fill field)
              DS    H            to fill the gap
```

TODR Time of day register format.
The data area for accepting the timespan has the following format (macro expansion with MF=D and PREFIX=N):

```
NTICDT         DS    0XL8        tdiff in TODR (for C)
NTICDTMS       DS    F           most significant word:
*                                   approx. sec
NTICDTLS       DS    F           least significant word:
*                                   micro_sec * 2**12
```

TODX Extended time of day register format.
The data area for accepting the time stamp has the following format (macro expansion with MF=D and PREFIX=N):

```
NTICSX          DS    0XL8        timestamp in TODX
NTICSTHW        DS    F           high word
NTICSTLW        DS    F           low word
```

Value ranges

The following value ranges are valid for the **CTIME** macro. Note that the macro format distinguishes between input and output with regard to time stamps and timespans.

BINAR or ISO4 format:

```
01.01.1900 00:00:00,000000  <  time stamp  <  31.12.9999 23:59:59,999999
```

Permitted time difference: ± 2147483647 days

TODR format:

```
t0   <  time stamp  <  t1
```

Permitted time difference: ± 26062 days

`t0` and `t1` are dependent on the epoch which is set for the TOD register in the startup parameter service (parameter record GTIME, see the “Introduction to System Administration” manual [10]) for the system run. The TODR format for the default epoch (GTIME parameter EPOCH=00) is:

```
01.01.1900 00:00:00.000000  <  time stamp  <  17.09.2042 23:53:47.370495
```

TODX format:

```
01.01.1900 00:00:00  <  time stamp  <  18.03.4317 02:44:48.587775
```

Permitted time difference: ± 882867 days

Macro format and description of operands

CTIME
FUNCT=*CONV / *ADD / *ADDLL / *DIFF / addr / (r) ,BASE1IN=*UTC / *LTI / *FZ / addr / (r) ,FRM1IN=*ISO4 / *ISO4MIC / *BINAR / *BINARMIC / *TODR / *TODX / addr / (r) ,INF1IN=*CALEND / *JULIAN / addr / (r) ,FRM1ZIN=*NONE / *ISO4 / *ISO4LST / *BINAR / *BINARLST / addr / (r) ,CHDL1IN=*NONE / addr / (r) ,BASE2IN=*UTC / *LTI / *FZ / addr / (r) ,FRM2IN=*ISO4 / *ISO4MIC / *BINAR / *BINARMIC / *TODR / *TODX / addr / (r) ,INF2IN=*CALEND / *JULIAN / addr / (r) ,FRM2ZIN=*NONE / *ISO4 / *ISO4LST / *BINAR / *BINARLST / addr / (r) ,CHDL2IN=*NONE / addr / (r) ,BASEOUT=*LTI / *UTC / *FZ / addr / (r) ,FRMOUT=*ISO4 / *ISO4MIC / *BINAR / *BINARMIC / *TODR / *TODX / addr / (r) ,FRMZOUT=*NONE / *ISO4 / *ISO4LST / *BINAR / *BINARLST / addr / (r) ,CHDLOUT=*NONE / addr / (r) ,FRMDIN=*ISO4 / *ISO4MIC / *BINAR / *BINARMIC / *TODR / *TODX / addr / (r) ,FRMDOUT=*ISO4 / *ISO4MIC / *BINAR / *BINARMIC / *TODR / addr / (r) ,LINKADR=*NONE / linkaddr ,MF=D / C / L / M / E [,PARAM=addr / (r)] ,PREFIX=N / p ,MACID=TIC / macid

The operands are described in alphabetical order below.

BASE1IN=
BASE2IN=
BASEOUT=
BASE1IN and BASE2IN specify the time base valid for the 1st and 2nd input time stamps and BASEOUT specifies the time base valid for the output time stamp.

***UTC**
The time stamp is specified according to UTC (Universal Time Coordinate, which corresponds to Greenwich Mean Time).
This is the default value for BASE2IN.

***LTI**
The time stamp is specified according to LTI (Local TIme), the local time base valid in the calling system. This is the default value for BASEOUT.

***FZ**
The time stamp is specified in a foreign or any time base (Foreign Zone). The information that characterizes this time base must be included in the zone information for the time stamp involved and specified with the relevant parameters (the FRMxZIN or FRMZOUT operand must be specified).

addr
Symbolic address (name) of a field that must be loaded beforehand with the relevant equate. Length = 1 byte.

(r)
Register containing the value of the relevant equate.

CHDL1IN=
CHDL2IN=
CHDLOUT=
Provides a table (list) of conversion times for use in assigning FZ time stamps to summer time or winter time.
A specification other than *NONE is only permissible with MF=M and with “FZ” time stamps.

<u>*NONE</u>
No table provided.

addr
Symbolic address (name) of a field containing the table of conversion times. This operand value is only permitted with MF=M and may only be specified in conjunction with BASExxx=*FZ. To process this table, the values FRMxZIN=*ISO4LST/*BINARLST or FRMZOUT=*ISO4LST/*BINARLST must be set. For unique representation of a time stamp, the zone information fields zone difference and conversion difference must also be assigned.
The table must be aligned on a word boundary and the format must be as follows:

The conversion time information is contained in consecutive doublewords. CHDATE data appears as UTC-based STCK values shifted 8 bits logically to the right (SRL command), with the lowest-value bit indicating the nature of the conversion: if the bit = 0, conversion is from winter to summer time, if the bit = 1, conversion is from summer to winter time.
The table should be generated with **GTIME** CHDATE=...
These doublewords must contain ascending values.
The table must end with the doubleword D'0'.
See page 372 for a worked example.

(r)
Register with the address value of addr.

FRMDIN=
FRMDOUT
Specifies the existing timespan format (FRMDIN as input timespan) or the desired timespan format (FRMDOUT as output timespan).

***ISO4**
The timespan is represented in printable format. With FRMDOUT, millisecond and microsecond fields are also provided.

***ISO4MIC**
The timespan is represented in printable format. With FRMDIN, additional expansion with milliseconds and microseconds is expected.

***BINAR**
The timespan is represented in binary form with fixed-point numbers. 4 bytes are used for the days and 2 bytes each for the hours, minutes and seconds. With FRMDOUT, millisecond and microsecond fields are also provided.

***BINARMIC**
The timespan is represented in binary form with fixed point numbers. With FRMDIN, additional expansion with milliseconds and microseconds is expected.

***TODR**
The timespan is represented in time of day register format.

***TODX**
The timespan is represented in extended time of day register format.

addr
Symbolic address (name) of a field that must be loaded beforehand with the relevant equate. Length = 1 byte.

(r)
Register containing the value of the relevant equate.

FRM1IN=
FRM2IN=
FRMOUT=
Specifies the existing time stamp format (FRM1IN, FRM2IN as input time stamps) or the desired time stamp format (FRMOUT as output time stamp).

***ISO4**
The time stamp is represented in printable format. This is the default value for FRM2IN and FRMOUT. With FRMDOUT, millisecond and microsecond fields are also provided.

***ISO4MIC**
The time stamp is represented in printable format. With input time stamps, additional expansion with milliseconds and microseconds is expected.

***BINAR**
The time stamp is represented in binary form with halfword fixed-point numbers. With FRMDOUT, millisecond and microsecond fields are also provided.

***BINARMIC**
The time stamp is represented in binary form with halfword fixed point numbers.
With input time stamps, additional expansion with milliseconds and microseconds is expected.

***TODR**
The time stamp is represented in time of day register format.

***TODX**
The timespan is represented in extended time of day register format.

addr
Symbolic address (name) of a field that must be loaded beforehand with the relevant equate. Length = 1 byte.

(r)
Register containing the value of the relevant equate.

FRM1ZIN=
FRM2ZIN=
FRMZOUT=
Specifies the format of the additional zone information supplied (FRMxZIN) or desired (FRMZOUT), which characterizes the time base of the specified time stamp.
This operand is mandatory if the time base “FZ” was specified for the time stamp involved.

If the “LTI” time base was specified for an input time stamp, it is possible to specify whether or not the season information supplied with the time stamp is to be evaluated. “*NONE” means that this information is not evaluated. Other values require the corresponding field of the zone information to contain a valid value.

For the output time stamp, the FRMZOUT operand specifies the format in which the zone information is to be output. If “*NONE” is specified, the format of the time stamp itself is assumed (if the “*TODR” format is used, the zone information is output in binary form).

<u>*NONE</u>
No zone information is supplied.

***ISO4**
The zone information is specified in printable format.

***ISO4LST**
The zone information is specified in printable format. The address of the CHDATE table must be provided by the operands CHDLxIN or CHDLOUT.
The specification in the zone information which shows whether the time stamp is for summer or winter time is ignored. The FZ time stamp is compared with this table to obtain the summer / winter time information (see also the example on page 372).

***BINAR**
The zone information is specified in binary form as a halfword fixed-point number.

***BINARLST**
The zone information is specified in binary form as a halfword fixed-point number.
The address of the CHDATE table must be provided by the operands CHDLxIN or CHDLOUT.
The specification in the zone information which shows whether the time stamp is for summer or winter time is ignored. The FZ time stamp is compared with this table to obtain the summer / winter time information (see also the example on page 372).

addr
Symbolic address (name) of a field that must be loaded beforehand with the relevant equate. Length = 1 byte.

(r)
Register containing the value of the relevant equate.

FUNCT=
Specifies which function of the **CTIME** macro is to be executed.

***CONV**
The 1st input time stamp is converted into a different format. The results are written to the output time stamp.

***ADD**
The input timespan is added to the 1st input time stamp. The results appear in the output time stamp. The input timespan may also be negative. The format of the input timespan is specified with the FRMDIN operand.

***ADDLL**
As is the case with *ADD, the input timespan is added to the 1st input time stamp.

Here, however, a conversion between summer time and winter time is taken into account between the input time stamp and the output time stamp. (These conversion days do not have the usual length of 24 hours; they are lengthened or shortened by the conversion difference. When addition is carried out, however, they are nevertheless considered to be 24-hour days, in contrast to *ADD). The results appear in the output time stamp. The input timespan can also be negative.

Example
The conversion from winter to summer time took place on 30.03.2008 at 02:00:00.

```
Input time stamp        2008-03-29,23:00:00
Input timespan              +00001-00:00:00 (1 Tag)

*ADD result             2008-03-31,00:00:00
*ADDLL result           2008-03-30,23:00:00
```

***DIFF**
The time difference between the 1st input time stamp and the 2nd input time stamp is calculated. The results appear in the output timespan.

addr
Symbolic address (name) of a field that must be loaded beforehand with the relevant equate. Length = 1 byte.

(r)
Register containing the value of the relevant equate.

INF1IN=
INF2IN=
Specifies the type of day specification used in the input time stamp involved.

<u>*CALEND</u>
The conventional calendar month and day specification is used.

***JULIAN**
The date is specified according to the Julian system, which numbers the days of the year consecutively, starting at January 1.

addr
Symbolic address (name) of a field that must be loaded beforehand with the relevant equate. Length = 1 byte.

(r)
Register containing the value of the relevant equate.

LINKADR=
Specifies the manner in which the address of the entry point IGTCTI for the **CTIME** routine in the GET-TIME subsystem is supplied to the user program. LINKADR must be specified if MF=E; in all other cases specifying LINKADR has no effect.

NONE

During assembly the assembler generates an external reference for the IGTCTI entry point and this is resolved during linking via the autolink function of the BLS.
This value can be used when the module containing the **CTIME** call

– is always linked and loaded with the dynamic binder loader DBL (in this case, **CTIME** in the E form is allowed to issue a V constant, which is supplied by the BLS during the load procedure) or
– is linked with the BINDER of the new BLS (see the “BINDER” manual [5] under the BINDER statement SET-EXTERN-RESOLUTION RESOLUTION=STD.

linkaddr

Symbolic address (name) of a word in which the user has provided the address of the IGTCTI entry point before the **CTIME** call.
The following example shows how the address of the IGTCTI entry point is first supplied to the program by an appropriate **BIND** call in register R1 and can then be transferred for the **CTIME** call into the word designated by “linkaddr”:

```
          BIND   MF=E,PARAM=BINDPL
          :
          CTIME  MF=E,PARAM=OPLIST,LINKADR=AENTRY
          :
AENTRY    DS     F
OPLIST    CTIME  MF=L,...
BINDPL    BIND   MF=L,SYMBOL=IGTCTI,SYMBLAD=AENTRY
```

The entry address of the **CTIME** routine must always be supplied to the user program in this way if none of the cases mentioned in LINKADR=*NONE apply, e.g. especially if the module with the **CTIME** call is linked by BINDER under the BINDER statement SET-EXTERN-RESOLUTION RESOLUTION=MANDATORY.

MF=

For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

In the E form and M form of the macro, the label of the data area is specified in the PARAM operand. The data area must be doubleword-aligned. Default setting: NTICPA
aligned. Default setting: NTICPA
When calling the macro with MF=L, the user must specify this label explicitly, otherwise an MNOTE is output.

Use of registers

The following registers are required for a **CTIME** macro call:

R1	is loaded by the macro with the address of the operand list.
R13	must be loaded before the macro call with the address of an 18-word save area which the calling program has to provide.
R14	is loaded by the macro with the return address of the user program.
R15	is overwritten by the routine called (via **CTIME**).

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to execution of the CTIME macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

cc	bb	aaaa	Meaning
00	00	0000	Function executed successfully
	01	0001	The function was aborted: an invalid time stamp was specified in the parameter list. *Action:* correct time stamp.
	01	0002	The function was aborted: error in the zone information of a specified time stamp. *Action:* correct zone information.
	01	0003	The function was aborted: error in timespan specification. *Action:* correct timespan specification.
	01	0006	The function was aborted: error in specification of the input or output data via the operands in the parameter list. *Action:* correct specification.
	01	000D	The function was aborted: address of CHDATEs table not aligned on word boundary or address field is empty. *Action:* correct program.
	01	000E	The function was aborted: error in format of CHDATEs table. The desired data processing is not possible. Possible errors: 1. The first byte of each entry is not X'00' . 2. The last byte of each entry is not alternately X'00' and X'01'. 3. The entries are not monotonously ascending in terms of the time stamp for assignment. This may be because – the table was wrongly formated with GTIME; – e.g. CHDATEs were only entered for the next two years but a three-year time stamp is to be assigned; the data following the table is in this case taken to be a continuation of the table. 4. The time difference between two entries (excluding the first two) is not between 4 to 8 months *Action:* correct table or table-formation program.
00	04	FFFF	The function was aborted: Data area is not aligned on a double word boundary. *Action:* correct program.
02	00	0007	Warning: the function was executed but the output time stamp is before or after the conversion times. It is not possible to determine whether summer or winter time applies. It is assumed that a winter time stamp is required. To make sure that the output time stamp is unambiguous, zone information must be stored with it. This warning can occur only for an LTI time stamp or for an FZ time stamp for which a CHDATE table has been provided.

cc	bb	aaaa	Meaning
02		0008	Warning: the function was executed but the specified time stamp is represented in a conversion time interval that cannot actually occur. A winter time stamp is assumed. If it is known whether summer or winter time applies, this information can be supplied with the time stamp (see FRMnZIN operand). This warning can occur only for an LTI input time stamp or for an FZ time stamp for which a CHDATE table has been provided. *Action:* if information is available as to whether summer or winter time applies, it can be supplied with the time stamp. To process this information, the FRMxZIN operand must be set accordingly.
02		0009	Warning: input time stamp is not unique. The function was executed but the specified time stamp is now represented in a conversion interval in which the time stamp is represented twice. A summer time stamp is assumed. If it is known whether the time stamp is a summer or winter time stamp, this information can be supplied with the time stamp (see FRMnZIN operand). This warning can occur only for an LTI input time stamp or for an FZ time stamp for which a CHDATE table has been provided. *Action:* if information is available as to whether summer or winter time applies, it can be supplied with the time stamp. To process this information, the FRMxZIN operand must be set accordingly.
02		000A	Warning: output time stamp is not unique. The function was executed but the output time stamp is in a time interval at the conversion between summer and winter time that occurs twice. To ensure that the output time stamp is unambiguous, the zone information must be stored with the time stamp. This warning can occur only for an LTI output time stamp or for an FZ time stamp for which a CHDATE table has been provided.
02		000B	Warning: the function was executed but the upper range limit (see “Value ranges”) was exceeded when a timespan was added to a time stamp. The output date receives the value of the upper range limit.
02		000C	Warning: the function was executed but the result of subtracting a timespan from a time stamp was a value less than the lower range limit (see “Value ranges”). The output date receives the value of the lower range limit.
02		000F	Warning: the function was executed but the output time stamp is in a conversion time interval which is not actually permitted. This warning can occur only for *ADDLL function output time stamps.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

If subcode1 is not equal to zero, only the return code in the standard header is set and no other data is transfered.

The calling program is terminated when the following errors occur:

- The data area is not assigned to the caller.
- The data area is not aligned on a doubleword boundary.
- The data area is protected against write access.

Example illustrating utilization of a user-specific CHDATEs table

```
         TITLE 'GTIME and CTIME'
*
         PRINT NOGEN,BASE
         GPARMOD 31
*
CGTIME   @ENTR TYP=M
         BIND MF=E,PARAM=BINDPLG
         BIND MF=E,PARAM=BINDPLC
*
         LA    R1,GTPAR1
         USING NTIGPL,R1
         MVC   NTIGCHD(NTIGCHDL),=A(0,128)                                (1)
         LA    R3,CHDATES                                                 (2)
*
         @CYCL ,
         GTIME MF=E,PARAM=(R1),LINKADR=AENTRYG                            (3)
         MVC   0(NTIGCHDL,R3),NTIGCHD                                     (4)
         LA    R3,NTIGCHDL(R3)                                            (5)
*
         @WHEN NE
         CLC NTIGRET,=A(0)
         @BREA ,                                                          (6)
*
         @BEND ,
*
**********************************************************************
* Following this part of the program, the system's CHDATES are       *
* already stored in the CHDATES memory area.                         *
**********************************************************************
*
DTH1     LA    R1,CTPAR2
         USING NTICPL,R1
         CTIME MF=M,CHDLOUT=CHDATES
*
         RDATA MF=(E,READDATE)                                            (7)
         LA    R1,CTPAR2
         USING NTICPL,R1
         MVC   NTIC1MD(L'DATEDATE),DATEDATE                               (8)
         MVC   NTIC1MT(L'DATETIME),DATETIME
         MVC   NTIC3MZ(14),=C'+01:00:-01:00-'
         CTIME MF=E,PARAM=(R1),LINKADR=AENTRYC ------------------------  (9)
*
END      @EXIT
*
**********************************************************************
* With the above section of the program, RDATA is used to read in a  *
* date in the format "yyyy mm dd hh mm ss" from the terminal and to  *
* convert it using the CTIME from UTC to FZ, taking into account the *
* CHDATEs table. The event is shown in ISO4 format in the parameter  *
* list in the output time stamp.                                     *
**********************************************************************
*
READDATE RDATA INDATE,0,MF=L ------------------------------------------ (10)
*
INDATE   DS    0CL30
DATELEN  DS    CL2
DATERES  DS    CL2
DATEDATE DS    CL10
DATESPAC DS    CL1
DATETIME DS    CL8
*
GTPAR1   GTIME MF=L,CHDATE=NEXT ---------------------------------------(3)
*
CTPAR2   CTIME MF=L,FUNCT=*CONV,BASE1IN=*UTC,FRM1IN=*ISO4,            C
               BASEOUT=*LTI,FRMOUT=*ISO4 ------------------------------ (9)
*
CHDATES  DS    100D ----------------------------------------------------- (11)
*
BINDPLG  BIND MF=L,SYMBOL=I@GTIME,SYMBLAD=AENTRYG
BINDPLC  BIND MF=L,SYMBOL=IGTCTI,SYMBLAD=AENTRYC
*
AENTRYG  DS    A
AENTRYC  DS    A
*
         @END   ,
**********************************************************************
* DSECTs                                                             *
**********************************************************************
*
NTICPL   CTIME MF=D
NTIGPL   GTIME MF=D
         END
```

(1) Default setting of the **GTIME** parameter list with a value that ensures that the subsequent CHDATEs are supplied in chronological order.

(2) The storage address of the CHDATE is loaded in register R3.

(3) The **GTIME** call fetches the first or next CHDATE.

(4) Secured in the storage area.

(5) The pointer to the CHDATEs table is incremented by X'08'.

(6) The loop should be quit if there is no next CHDATE (or in the event of an error).

(7) The **RDATA** requires that a date be entered via a data display terminal. The date must be entered in the format `yyyy-mm-dd hh:mm:ss` (it is not necessary to enter the divider; it is sufficient to enter a blank in each case).

(8) The **CTIME** data area is supplied with date and zone.

(9) The input time stamp is converted from UTC to LTI. Both the output and input time stamps should be in ISO4 format.

(10) **RDATA** data area with definition of the entry field `INDATE`.

(11) Area definition for up to 100 CHDATEs.

Runtime log

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,cgtime), -
//        compiler-action=module-generation(module-format=llm), -
//        macro-library=$tsos.syslib.assembh.012, -  ——————————————————— (12)
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,cgtime)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 1605 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 101 MSEC
//end
%  ASS6012 END OF ASSEMBH
/add-file-link link-name=blslib00,file-name=$tsos.syslib.assembh.012 —— (13)
/load-executable-program library=macexmp.lib,element-or-symbol=cgtime, -
/     dbl-parameters=*par(resolution=*par(alternate-libraries=*yes))
%  BLS0523 ELEMENT 'CGTIME', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'CGTIME', VERSION ' ' OF '<date> <time>' LOADED
/%in dth1 <%d %@(chdates) -> %xl96>  ———————————————————————————————— (14)
/%in end <%d indate,%1 -> %xl232> ——————————————————————————————————— (15)
/%r
```

```
*** TID: 005000D8 *** TSN: 2QSE ******************************************
CURRENT PC: 0000006A    CSECT: CGTIME  *************************************
V'00000208' = CGTIME   + #'00000208'  ----------------------------------- (16)
00000208 (00000208) 008FF960 489C4000 0090D566 AC464001    ..9-.. ...N... .
00000218 (00000218) 0091BA3A 1E2A4000 00929F0D 900E4001    .j.... ..k.... .
00000228 (00000228) 009383E1 01F24000 009468B4 73D64001    .lc..2 ..m...O .
00000238 (00000238) 00954D87 E5BA4000 0096325B 579E4001    .n(gV. ..o.$.. .
00000248 (00000248) 0097172E C9824000 009804CF 49A04001    .p..Ib ..q.... .
00000258 (00000258) 00FFFFFF FFFFFF00 0099CE76 2D684001    .~~~~~~..r.... .
*2012-01-20 14:36:35 ------------------------------------------------------ (17)
SRC_REF:   230 SOURCE: CGTIME  PROC: CGTIME  ******************************
INDATE         = |....2012-01-20 14:36:35.......|
CURRENT PC: 00000098    CSECT: CGTIME  *************************************
V'00000120' = CGTIME   + #'00000120' ------------------------------------ (18)
00000120 (00000120) 00050702 00000000 01010001 01010001    ................
00000130 (00000130) 02010000 01010100 F2F0F1F2 60F0F160    ........2012-01-
00000140 (00000140) F2F0F0F0 F0404040 F1F47AF3 F67AF3F5    20000   14:36:35
00000150 (00000150) 00000000 00000000 00000000 00000000    ................
00000160 (00000160) 00000000 00000000 F0F0F0F0 60F0F060    ........0000-00-
00000170 (00000170) F0F0F0F0 F0404040 F0F07AF0 F07AF0F0    00000   00:00:00
00000180 (00000180) 00000000 00000000 00000000 00000000    ................
00000190 (00000190) 00000000 00000000 4EF0F0F0 F0F0F0F0    ........+0000000
000001A0 (000001A0) F0F0F060 F0F07AF0 F07AF0F0 00000000    000-00:00:00....
000001B0 (000001B0) 00000000 00000000 F2F0F1F2 60F0F160    ........2012-01-
000001C0 (000001C0) F2F0F0F2 F040C6D9 F1F57AF3 F67AF3F5    20020 FR15:36:35
000001D0 (000001D0) 4EF0F17A F0F060F0 F17AF0F0 60E6F0F0    +01:00-01:00-W00
000001E0 (000001E0) F0F0F0F0 00000208 00000000 0000016C    0000...........%
000001F0 (000001F0) 00000000 00000000 00000000 00000000    ................
00000200 (00000200) 00000000 00F8C180                      .....8A.
```

(12) During assembly, the library containing the necessary @ macros for structured programs is specified.

(13) In order to run structured programs, the library with the @ macros will also be required. Here it is assigned in the load call using the link name BLSLIB00 and the specification ALTERNATE-LIBRARIES=*YES.

(14) The CHKDATEs table is to be output at the symbolic address DTH1.

(15) The INDATE field and the operand list are to be outpout at the symbolic address END. After a successful **CTIME** call, the start address of the operand list is contained in register R1. The operand list is 232 bytes in length (this information is obtained by releasing the DSECT with **CTIME** MF=D).

(16) The symbolic address DTH1 has been reached. The table of the CHDATEs is output. 10 CHDATES are currently recognized in the system. They are in TODR format, shifted 1 byte to the right.
The system marks the end of the CHDATEs table with X'00FFFFFF FFFFFF00'.

(17) Entry of a time stamp is requested (Prompt “*”).

(18) The symbolic address END has been reached. The contents of the field INDATE are output: The time stamp entered was read in correctly by **RDATA**.

Output of the operands list:

- The first input time stamp of the operand list corresponds to the time stamp “2012-01-20 14:36:35” read in by **RDATA**. The time basis is UTC. The time stamp is in ISO4 format.
- The second input time stamp and the input timespan was not specified.
- The output time stamp (based on LTI) is in ISO4 format and contains the following information:
  The day specified is the 20th day of the year, a Friday.
  The current time has been recalculated. The LTI time is “15:36:35”.
  The zone information “+01:00-01:00-W” shows that the zone difference between UTC and LTI is one hour and the difference between daylight saving time and standard time is also one hour, and that now is standard time (“W” as “winter“, standard time; summer time would be indicated by “S” as “summer”).
- No output timespan was calculated.

CUPAB – Address operand list (24-bit interface)

General

Application areas:	Requesting and accessing lists and tables; see page 155 Input/output; see page 156
Macro type:	Type O; see page 28

The description applies to TIAM V13.2A.

The **CUPAB** macro may be used only in 24-bit addressing mode.
In 31-bit addressing mode, the form MF=F/D of the appropriate input/output macro (**RDATA**, **WROUT**, **WRTRD**) must be used.

Macro description

The **CUPAB** macro (Communication User PArameter Block) macro allows the user to address symbolically the fields and flags in the operand lists for the **RDATA**, **WROUT** and **WRTRD** macros. For this purpose, **CUPAB** generates a dummy section (DSECT) for the operand list with 24-bit addresses.

Macro format and description of operands

[name] CUPAB
D

name
When this entry is specified, it is used as the DSECT name. When it is omitted, CUPAB is automatically generated as the DSECT name.

D
D generates a dummy section (DSECT) for the operand list. When this operand is omitted, an MNOTE message is issued.
DSECT generation is not affected.

The field names generated by the macro and their characteristics are as follows:

- **RDATA** operand table:

Field Name	Byte	Meaning
CURAREAW	0 - 3	Full word containing CUREDIT1 and CURAREA.
CUREDIT1	0	Input edit option byte 1.
CURAREA	1 - 3	Address of user input area (in CURAREAW).
CURFTB	4	Flag.
CUREDIT2	5	Input edit option byte 2.
CURALEN	6 - 7	Length of user input area.
CURERRW	8 - 11	Full word containing CURACI and CURERROR.
CURACI	8	Assignment change indicator for SYSDTA
CURERROR	9 - 11	Error address (in CURERRW).
L@RDATAB		Length of RDATA operand table.

- **WROUT** operand table:

Field Name	Byte	Meaning
CUWMSGW	0 - 3	Full word containing CUWEDIT1 and CUWMSG.
CUWEDIT1	0	Output edit option byte 1.
CUWMSG	1 - 3	Address of the message in the user program.
CUWERRW	4 - 7	Full word containing CUWEDIT2 and CUWERROR.
CUWEDIT2	4	Output edit option byte 2.
CUWERROR	5 - 7	Error address.
L@WROUTB		Length of the WROUT operand table.

- **WRTRD** operand table:

Field Name	Byte	Meaning
CUBMSGW	0 - 3	Full word containing CUBOEDT1 and CUBMSG.
CUBOEDT1	0	Output edit option byte 1.
CUBMSG	1 - 3	Address of message output area.
CUBAREAW	4 - 7	Full word containing CUBIEDT1 and CUBAREA.
CUBIEDT1	4	Input edit option byte 1.
CUBAREA	5 - 7	Input area address.
CUBOEDT2	8	Output edit option byte 2.
CUBIEDT2	9	Input edit option byte 2.
CUBALEN	10 - 11	Length of user input area.
CUBERRW	12 - 15	Full word containing CUBERROR.
reserviert	12	-
CUBERROR	13 - 15	Error address.
L@WRTRDB		Length of the WRTRD operand table.

Symbolic constants for edit bytes 1 and 2

As well as defining field names for the **RDATA, WROUT** and **WRTRD** operand tables, **CUPAB** also defines symbolic constants for the values of the edit, edit1 and edit2 operands.

The following tables provide an overview of the names of the symbolic constants defined by **CUPAB**, their current values and equivalent symbolic edit operands when MODE is specified, and also indicate whether or not they are valid in the various modes.

a) Output edit option byte 1

CUPAB name	Bit	Corresponding MODE operand	valid (X) or mandatory (1) for MODE=			
			COMP	LINE	FORM	PHYS
CWR1CODE	2^0	OTRSUP=	X	0	0	0
CWR1LNET	2^1	OLINEND=	X	0	1	1
	2^2	reserved for MODE=	0	1	0	1
CWR1RSET	2^3	OMANUAL=	X	0	0	0
CWR1HOM	2^4	OHOM=	0	X	0	0
CWR1PTPE	2^5	OPTAPE=	X	0	0	0
	2^6	reserved for MODE=	0	0	1	1
CWR1HARD	2^7	OHCOPY=	X	X	0	X

b) Output edit option byte 2

CUPAB name	Bit	Corresponding MODE operand	valid (X) or mandatory (1) for MODE=			
			COMP	LINE	FORM	PHYS
CWR2HDR	2^0	OHDR=	X	0	1	X
CWR2NOLC	2^1	ONOLOGC=	0	X	0	0
CWR2EXT	2^2	EXTEND=	0	X	0	0
CWR2INFO	2^3	OINFO=	0	X	0	0
	2^4	reserved	0	0	0	0
CWR2POSN	2^5	ONOPOSN=	0	X	0	0
CWR2TRAN	2^5	OTRANS=	0	0	0	X
CWR2BEL	2^6	OBELL=	0	X	0	0
CWR2ETB	2^7	OETB=	0	0	0	X

c) Input edit option byte 1)

CUPAB name	Bit	Corresponding MODE operand	valid (X) or mandatory (1) for MODE=			
			COMP	LINE	FORM	PHYS
CRD1CODE	2^0	ITRSUP=	X	0	0	X
CRD1LNET	2^1	ILINEND=	X	0	1	1
CRD1BACK	2^2	IGETBS=	X	X	X	X
CRD1RSET	2^3	IMANUAL=	X	0	0	0
CRD1LCT	2^4	ILCASE=	X	X	X	X
	2^5	reserved for MODE=	0	1	0	1
	2^6	reserved for MODE=	0	0	1	1
CRD1HDR	2^7	IHDR=	X	0	1	X

d) Input edit option byte 2

CUPAB name	Bit	Corresponding MODE operand	valid (X) or mandatory (1) for MODE=			
			COMP	LINE	FORM	PHYS
CRD2GFC	2^0	IGETFC=	0	X	0	0
	2^1	reserved	0	0	0	0
CRD2CFD	2^2	ICFD=	0	X	0	0
CRD2GIC	2^3	IGETIC=	0	X	0	0
	2^4	reserved	0	0	0	0
CRD2EXT	2^5	EXTEND=	0	X	0	1
	2^6	reserved	0	0	0	0
	2^7	reserved	0	0	0	0

The following applies to operands where bits are not reserved:

MODE operand	Associated bit
= Y	set (1)
= N	reset (0)

For the meaning of the MODE operands (and the associated bits of the input and output edit option bytes) refer to the operand descriptions for the **RDATA**, **WROUT** and **WRTRD** macros.

DCSTA – Generate operand table for terminal attributes

General

Application areas:	Requesting and accessing lists and tables; see page 155 Input/output; see page 156
Macro type:	Type O; see page 28

- The description applies to VTSU V13.3A

Macro description

The **DCSTA** macro (Data Communication STation Attributes) supports the **TSTAT** macro. **DCSTA** is used to generate receiving fields or symbolic field names (DSECT) for the information supplied by the **TSTAT** macro.

Macro format and description of operands

```
[name] DCSTA

D / C
[,prefix]
,TYPE=TCHAR / PHDIM / LIDIM / VDT[YP] / EDOPT / OFLOW / STNAM / PRNAM / ALL / MONCS /
        PERPH / BASIC
```

name
Becomes the symbolic name for the first DS statement in the macro expansion if the C operand is specified (the length attribute is zero).
If the D operand is specified, the “name” operand specifies the name of the dummy section (DSECT).

If the “name” operand is omitted, the system forms a name from the valid prefix (see the “prefix” operand) and the operand value for TYPE.

C
A storage area with symbolic addresses is generated. In the **TSTAT** call, this area may be specified as receiving field. No CSECT statement is generated.

D
A dummy section (DSECT) is generated together with a DSECT statement. The symbolic name generated can be used to address a receiving field previously defined for the **TSTAT** macro.

prefix
The prefix may consist of 1 to 3 characters, which are to appear at the beginning of the generated field names. The specified prefix replaces the characters “STA”, the default value for the field name prefix.

TYPE=
Specifies which receiving field or which field names are to be generated:

TCHAR
Requests data terminal characteristics.

PHDIM
Requests physical data terminal characteristics.

LIDIM
Requests logical data terminal characteristics.

VDT[YP]
Requests logical data terminal type.

EDOPT
Static edit options.

OFLOW
Requests overflow control type values.

STNAM
Requests data terminal name.

PRNAM
Requests processor name.

ALL
Requests information from TCHAR to PRNAM.

MONCS
Description of monitor and character sets.

PERPH
Requests connected peripherals.

BASIC
Basic information on the data terminal.

Functional description

If the **TSTAT** macro is used to request information, the receiving field may be defined either explicitly or with the **DCSTA C,...** macro. If the receiving field has been explicitly defined, the **DCSTA D,...** macro may be used to generate a dummy section (DSECT), i.e. an address structure for the receiving field, by means of a USING statement. In order to interrogate bit values, the **DCSTA** macro generates symbolic constants with which the field contents may be compared.

The field names which **DCSTA** generates by default may be seen in the macro expansion in the example. All of them start with the letters STA. Any string may be selected to replace these characters.

The start addresses of the receiving fields generated by the **DCSTA C,...** macro must be specified in the **TSTAT** macro.

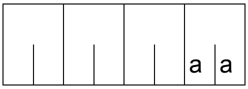
The default start addresses are as follows:

STATCHAR
STAPHDIM
STALIDIM
STAVDT
STAEDOPT
STAOFLOW

STASTNAM
STAPRNAM
STAALL
STAMONCS
STAPERPH
STABASIC

Return information and error flags

R15:



A return code relating to the execution of the macro DCSTA is transferred in the rightmost byte of register R15.

Return code	Meaning
X'00'	Normal termination
X'04'	Unrecoverable error
X'08'	Error in the operand list
X'0C'	No participant data terminal available
X'10'	Length of the receiving field is too short: only part of the information was given when the operand ALL was output. In all other operands no information is transferred.
X'14'	The desired information is (partly) not available

Description of the transferred information: see the following pages

- TCHAR:Physical type (area length: 8 bytes)

Byte	Symb. name	Meaning
0	STAPTTYP	Partner type:
	(STADCAMP) (STADCAMT)	Partner is a DCAM program Partner is a data terminal
1	STADVTYP	Device type, e.g.:
	(STAD1000) (STAD100E) (STADT100) (STADPT80) (STAD8110) (STAD8151) (STAD8152) (STAD8160) (STAD8162) (STAD9731) (STAD9750) (STAD9751) (STAD9752) (STAD9753) (STAD9754) (STAD9755) (STAD9763) (STAD8122) (STAD8121) (STAD9001) (STAD9002) (STAD9003) (STAD9004) (STAD9012) (STAD9013) (STAD0131) (STAD0189) (STAD9022) (STAD1118) (STAD1119) (STAD3270) (STADHOST) (STADAP) (STAD9021) (STAD3287) (STAD9014) (STAD9026) (STADFE)	T1000 Printer Terminal FS100-E Printer Terminal T100 Printer Terminal PT80 Printer Terminal 8110 Printer Terminal 8151 Data Display Terminal 8152 Data Display Terminal 8160 Data Display Terminal 8162 Data Display Terminal 9731 Graphics Workstation 9750/9749 Data Display Terminal 9751 Data Display Terminal 9752 Data Display Terminal 9753 Data Display Terminal 9754 Data Display Terminal 9755 Data Display Terminal 9763 Data Display Terminal 8122 Printer 8121 Printer 9001 Printer 9002 Printer 9003 Printer 9004 Printer 9012 Printer 9013 Printer 9001-31 Printer 9001-8931 Printer 9022 Printer 9011-18 Printer 9011-19 Printer 3270 Data Display Terminal Program in the server Workstation 9021 Printer 3287 Printer 9014 Printer 9026 Printer (HDLC, 9025 compatible) Front-End Data Display Terminal (FHS-DOORS)

Byte	Symb. name	Meaning
2	STATCHR2	Character set:
	(STATC2EX) (STATC2LC) (STATC2DT) (STATC2DF)	Second character set available Lowercase notation available German (instead of international) keyboard generated Byte 2 is defined
3	STATCHR3	Device options at the data display terminal:
	(STATC3H1) (STATC3H2) (STATC3IC) (STATC3AP) (STATC3GF) (STATC3DZ) (STATC3DF)	Local hardcopy printer generated or assigned with TCHNG Central hardcopy printer generated or assigned with TCHNG ID card reader generated or assigned with TCHNG APL option generated or assigned with TCHNG Graphics option generated or assigned with TCHNG Decentralized formatting Byte 3 is defined
4	STATCHR4	Data display terminal functions
	(STATC4CO) (STATC4ZF) (STATC4ST) (STATC4HI) (STATC4C8) (STATC4HP) (STATC4DF)	4 colors Character and field attributes Status of the terminal possible Hardware system line available 8 colors HP Laser Jet II Byte 4 is defined
5	STATTCHRS	Information from the status message
	(STATTCSDT) (STATTCSHC) (STATTCSIC) (STATTCSDF)	German keyboard connected Local hardcopy unit connected ID card reader connected Status of the terminal available
6	STACTRLU	Type of printer controller when the data terminal is a printer, X' 00 for 8112 Printer Controller, generated device type (see above) for data display terminals
7	STACHCAD	Channel address of the central hardcopy unit

Notes

- The 9749 Data Display Terminal can be generated as a separate device type in the PDN. However, for application programs, it is always displayed as the 9750 Data Display Terminal when the **TSTAT** macro is used.
- The 9758 M4 Data Display Terminal can be generated in the PDN as the 9755 or 9763 Data Display Terminal. However, for application programs, it is always displayed as the 9755 Data Display Terminal when the **TSTAT** macro is used.

- PHDIM:Physical attributes (area length: 8 bytes)

Byte	Symb. name	Meaning
0-1	STALLEN	Physical line length
2-3	STANOLIN	Physical line number (unrestricted for printer terminals: X' 7FFF')
4-5	STAMAXDB	Maximum physical device buffer, i.e. maximum number of characters which can be sent to the data terminal with one output call. X' 7FFF' (unrestricted): Restricted only by access method or line.
6-7	-	Reserved

Bit 2^{15} = 1 means in each case: value is not available.

- LIDIM:Logical attributes (line mode) (area length: 8 bytes)

Byte	Symb. name	Meaning
0-1	STALLLEN	Number of characters per physical line in line mode, where ' NL' in the text is to be counted as two characters.
2-3	STALNOLN	Number of physical lines which can be output in line mode without the overflow control reacting.
4-5	STALMAXB	Number of characters which can be sent in a message in line mode without the overflow control reacting (generally lines multiplied by columns minus 1).
6-7	-	reserved

Bit 2^{15} = 1 means in each case: value is not available.

- VDT[YP]:Logical type (area length: 8 bytes)

Byte	Symb. name	Meaning
0	STAVDT	Logical device type:
	(STALINCP) (STAFORCP) (STACMPCP) (STAFYSCP) (STAEXLCP) (STAAUTLF) (STANOINP) (STAEOM=0) (STAEOM=1)	Lines/page data terminal (LINE MODE) Format data terminal (FORM MODE) Data display terminal support compatible with earlier versions of operating system (COMP MODE) Physical data terminal support (PHYS MODE) Line/page data terminal (ext. LINE MODE) Automatic line feed for printers Data terminal is a printer Data terminal is a printer terminal Data terminal is a data display terminal
1	STAVDTPR	Logical device protocol:
	(STATD810) (STAT3270)	810 Protocol 3270 Protocol
2-7	-	reserved

- EDOPT:Static edit options (area length: 8 bytes)

The symbolic operands of the **WRTRD** macro are used for representation.

Byte	Symb. name	Meaning
0	STASEWR1	Output edit byte 1
	(STAWR1MM)	Mask for output edit mode =STAWR1CO: MODE=COMP =STAWR1LI: MODE=LINE =STAWR1FO: MODE=FORM =STAWR1FY: MODE=PHYS
	(STAWR1CD)	=1: OTRSUP =Y =0: =N
	(STAWR1LE)	=1: ONLINEND =Y =0: =N
	(STAWR1RE)	=1: OMANUAL =Y =0: =N
	(STAWR1HO)	=1: OHOM =Y =0: =N
	(STAWR1PT)	=1: OPTAPE =Y =0: =N
	(STAWR1HC)	=1: OHCOPY =Y =0: =N

Byte	Symb. name	Meaning
1	STASEWR2	Output edit byte 2
	(STAWR2HD)	=1: OHDR =Y
		=0: =N
	(STAWR2NO)	=1: ONOLOGC =Y
		=0: =N
	(STAWR2EX)	=1: EXTEND =Y
		=0: =N
	(STAWR2ET)	=1: OETB =Y
		=0: =N
	(STAWR2BL)	=1: OBELL =Y
		=0: =N
	(STAWR2TP)	=1: OTRANS =Y
		=0: =N
	(STAWR2IM)	=1: OINFO =Y
		=0: =N
	(STAWR2PN)	=1: ONOPOSN =Y
		=0: =N
2	STASERD1	Input edit byte 1
	(STARD1MM)	Mask for input edit mode
		=STARD1CO: MODE=COMP
		=STARD1LI: MODE=LINE
		=STARD1FO: MODE=FORM
		=STARD1FY: MODE=PHYS
	(STARD1CD)	=1: ITRSUP =Y
		=0: =N
	(STARD1LE)	=1: ILINEND =Y
		=0: =N
	(STARD1BS)	=1: IGETBS =Y
		=0: =N
	(STARD1PT)	=1: IMANUAL =Y
		=0: =N
	(STARD1LC)	=1: ILCASE =Y
		=0: =N
	(STARD1HD)	=1: IHDR =Y
		=0: =N
3	STASERD2	Input edit byte 2
	(STARD2FC)	=1: IGETFC =Y
		=0: =N
	(STARD2IC)	=1: IGETIC =Y
		=0: =N
	(STARD2CF)	=1: ICFD =Y
		=0: =N
	(STARD2EX)	=1: EXTEND =Y
		=0: =N

Byte	Symb. name	Meaning
4-7	-	reserved

- OFLOW:Control for screen overflow (area length: 8 bytes)

Byte	Symb. name	Meaning
0	STAOFLOW (STAOFCTM) (STAOFCAK) (STAOFCTL) (STAOFPGM)	Type of overflow control: Bit 2^0=1 Overflow control when using the timer. Waiting time with n seconds (hexadecimal input) according to STAOFTIM Bit 2^1=1 Overflow control with acknowledgement request when overflow occurs Bit 2^0=0 No overflow control Bit 2^1=0 No overflow control Bit 2^5=0 Overflow control by the system Bit 2^5=1 Overflow control by the user (see TCHNG macro, MODIFY-TERMINAL-OPTIONS command)
1	STAOFTIM	Waiting time (hexadecimal input in seconds).
2 - 7	-	reserved

- STNAM:Name of the terminal, as specified in the PDN (area length: 8 bytes)

Byte	Symb. name	Meaning
0-7	STASTNAM	Name of the terminal

- PRNAM:Name of the server to which the terminal is connected, as specified in the RDF generation (area length: 8 bytes)

Byte	Symb. name	Meaning
0-7	STAPRNAM	Name of the server

- ALL

The symbolic addresses of the table fields are defined as above. The subarea sequence is as follows:

TCHAR, PHDIM, LIDIM, VDTYP, EDOPT, OFLOW, STNAM, PRNAM

Note

> If TYPE=ALL is specified, the area size is 64 bytes. TYPE=ALL does not include MONCS, PERPH and BASIC.

- MONCS: Description of monitor and character sets (area length: at least 14 bytes)

Information is supplied in the status message from the data terminal, when this is available, otherwise default values are assumed.

Byte	Symb. name	Meaning
0	STAMOCPR	Status message from the terminal
	(STAMOCY) (STAMOCN)	Status message available. The following data is taken from the status message. No status message. VTSU default values are assumed for the following information.
1	STAMOTYP	Terminal monitor type
	(STAMONO) (STACOLOR) (STAPRINT)	Data terminal with a monochrome screen Data terminal with a color screen Data terminal is a printer
2	STAFAT	Field attributes
	(STAFATY) (STAFATN)	Field attributes can be used Field attributes cannot be used
3	-	reserved
4	STADIM1	Screen size 24 lines x 80 characters
	(STADIMY) (STADIMN)	Format is supported (can be addressed with DIM) Format is not supported (not addressable with DIM)
5	STADIM2	Screen size 32 lines x 80 characters
	(STADIMY) (STADIMN)	Format is supported (addressable with DIM) Format is not supported (not addressable with DIM)
6	STADIM3	Screen size 43 lines x 80 characters
	(STADIMY) (STADIMN)	Format is supported (addressable with DIM) Format is not supported (not addressable with DIM)
7	STADIM4	Screen size 27 lines x 132 characters
	(STADIMY) (STADIMN)	Format is supported (addressable with DIM) Format is not supported (not addressable with DIM)
8-11	-	reserved
12-13	STACSNO	Number of addressable character sets
14	STACS0T	Character set type 0
	(STACSSIN) (STACSTRI) (STACSNO)	Loadable monochrome character set Loadable color character set Nonloadable character set

Byte	Symb. name	Meaning
15	STACSOS (STACSNLO) (STACSDSS) (STACSDVN) (STACSDVA)	Character set status 0 Character set can be loaded Character set has been reserved by the data display terminal Character set has been loaded by the data processing system Character set has been loaded and assigned by the data processing system
16	STACS1T (STACSSIN) (STACSTRI) (STACSNO)	Character set type 1 Loadable monochrome character set Loadable color character set Nonloadable character set
17	STACS1S (STACSNLO) (STACSDSS) (STACSDVN) (STACSDVA)	Character set status 1 Character set can be loaded Character set has been reserved by the data display terminal Character set has been loaded by the processor Character set has been loaded and assigned by the processor
18	STACS2T (STACSSIN) (STACSTRI) (STACSNO)	Character set type 2 Loadable monochrome character set Loadable color character set Nonloadable character set
19	STACS2S (STACSNLO) (STACSDSS) (STACSDVN) (STACSDVA)	Character set status 2 Character set can be loaded Character set has been reserved by the data display terminal Character set has been loaded by the processor Character set has been loaded and assigned by the processor
20	STACS3T (STACSSIN) (STACSTRI) (STACSNO)	Character set type 3 Loadable monochrome character set Loadable color character set Nonloadable character set
21	STACS3S (STACSNLO) (STACSDSS) (STACSDVN) (STACSDVA)	Character set status 3 Character set can be loaded Character set has been reserved by the data display terminal Character set has been loaded by the processor Character set has been loaded and assigned by the processor
22	STACS4T (STACSSIN) (STACSTRI) (STACSNO)	Character set type 4 Loadable monochrome character set Loadable color character set Nonloadable character set

Byte	Symb. name	Meaning
23	STACS4S	Character set status 4
	(STACSNLO)	Character set can be loaded
	(STACSDSS)	Character set has been reserved by the data display terminal
	(STACSDVN)	Character set has been loaded by the processor
	(STACSDVA)	Character set has been loaded and assigned by the processor
24	STACS5T	Character set type 5
	(STACSSIN)	Loadable monochrome character set
	(STACSTRI)	Loadable color character set
	(STACSNO)	Nonloadable character set
25	STACS5S	Character set status 5
	(STACSNLO)	Character set can be loaded
	(STACSDSS)	Character set has been reserved by the data display terminal
	(STACSDSS)	Character set has been loaded by the processor
	(STACSDVN)	Character set has been loaded and assigned by the processor
	(STACSDVA)	
26	STACS6T	Character set type 6
	(STACSSIN)	Loadable monochrome character set
	(STACSTRI)	Loadable color character set
	(STACSNO)	Nonloadable character set
27	STACS6S	Character set status 6
	(STACSNLO)	Character set can be loaded
	(STACSDSS)	Character set has been reserved by the data display terminal
	(STACSDVN)	Character set has been loaded by the processor
	(STACSDVA)	Character set has been loaded and assigned by the processor
28	STACS7T	Character set type 7
	(STACSSIN)	Loadable monochrome character set
	(STACSTRI)	Loadable color character set
	(STACSNO)	Nonloadable character set
29	STACS7S	Character set status 7
	(STACSNLO)	Character set can be loaded
	(STACSDSS)	Character set has been reserved by the data display terminal
	(STACSDVN)	Character set has been loaded by the processor
	(STACSDVA)	Character set has been assigned and loaded by the processor

Bytes 14-29 are omitted when the length of the status area is inadequate, without giving a return code. Information on character sets is only given when the character sets can be addressed via bytes 12-13.

- PERPH: Connected peripherals (area length: 8 bytes)

Information is given in the status message from the data terminal, when this is available, otherwise it is supplied from the generation.

Byte	Symb. name	Meaning
0	STAPERPR (STAPERY) (STAPERN)	Status of the terminal Status message from the data terminal available No status message available from the data terminal
1-2	-	reserved
3	STALOCHC (STALHCY) (STALHCN)	Local hardcopy unit Local hardcopy unit connected No local hardcopy unit connected
4-5	-	reserved
6	STAIDCAR (STAIDCY) (STAIDCN)	ID card reader ID card reader connected No ID card reader connected
7	STACKT (STACKTY) (STACKTN)	Chipcard terminal Chipcard terminal connected No chipcard terminal connected

- BASIC: Basic information on the data terminal (area length: 24 bytes)

Information is given in the status message from the data terminal, when this is available, otherwise it is supplied from the generation or VTSU-B default values are assumed.

If you specify a length greater than 24 and less than 33 bytes, information is only returned for bytes 0-23.

If you specify a length of 33 bytes, information is only returned for bytes 0-32. If XHCS is not loaded or if the terminal does not support extended character sets, no information is supplied on the terminal type and the extended standard name.

If you specify a length greater than 33 bytes and less than 52 bytes and the data display terminal works in 8-bit mode, the first byte of this area is set to X'00'. The information received is truncated because the specified area is too small.

If you specify a length of 52 bytes, information is returned only for bytes 0-51. Bytes that are not required are set to X'00'. No information is provided for printers.

If you specify a length of 60 bytes, information is returned for bytes 0-59. Bytes that are not required are set to X'00'.

If you specify a length of 64 bytes, the information returned is complete (bytes 0-63).

Bytes that are not required are set to X'00'.

Byte	Symb. name	Meaning
0	STAINFO	Terminal status
	(STAINFOY)	Status message from the terminal is available
	(STAINFON)	No status message from the terminal
1	STAINFP	Status from the terminal
	(STAINFPY)	Status message from the terminal possible
	(STAINFPN)	No status message is expected from the terminal
2-9	STAPTNAM	Printable data terminal type
	TYP00	Unknown data terminal type
	DSS-X.29	X.29 data display terminal
	RECHNER	PROCESSOR
	SS-8102	8102 Printer
	DSS-8151	8151 Data Display Terminal
	DSS-8152	8152 Data Display Terminal
	SS-8110	8110 Teleprinter
	SS-8121	8121 Printer
	FS100	T100 Teleprinter
	FS100-E	FS100-E Teleprinter
	DRS90037	90037 Printer
	DRS-8122	8122 Printer
	DSS-8162	8162 Data Display Terminal
	DSS-8160	8160 Data Display Terminal
2-9	STAPTNAM	Printable data terminal type
	DRS-8124	8124 Printer
	AP	Application program
	SST-X.29	X.29 teleprinter
	DSS-9750	9750 or 9749 Data Display Terminal
	DRS-9003	9003 Printer
	DSS-9770	9770 Data Display Terminal
	DRS-9002	9002 Printer
	DSS-3974	3974 Data Display Terminal
	DSS-9751	9751 Data Display Terminal
	DSS-9752	9752 Data Display Terminal
	DSS-9753	9753 Data Display Terminal
	DRS-9001	9001 Printer
	DSS-9731	9731 Data Display Terminal
	DSS9770R	9770R Data Display Terminal
	DRS-9004	9004 Printer
	DSS-9754	9754 Data Display Terminal
	DSS-9755	9755 Data Display Terminal
	DSS-9763	9763 Data Display Terminal
	DRS-9012	9012 Printer

Byte	Symb. name	Meaning
2-9	STAPTNAM	Printable data terminal type
(cont.)	DRS-9013 DSS-3270 DRS-0131 DRS-0189 DRS-9022 DRS-1118 DRS-1119 DRS-3287 TCP-IP DRS-9021 DRS-9014 DRS-9026 DSS-FE	9013 Printer 3270 Data Display Terminal 9001-31 Printer 9001-8931 Printer 9022 Printer 9011-18 Printer 9011-19 Printer 3287 Printer TCP-IP application 9021 Printer 9014 Printer 9026 Printer (HDLC, 9025 compatible) Front-End Data Display Terminal (FHS-DOORS)
10	STAHCOPY	Local hardcopy unit
	(STABLHCY) (STABLHCN)	Local hardcopy unit connected No local hardcopy unit connected
11	STAIDCR	ID card reader
	(STAIDCRY) (STAIDCRN)	ID card reader connected No ID card reader connected
12	STACOL	Number of colors on the data terminal
	(STACOLNO) (STACOL4) (STACOL8)	No colors 4 colors 8 colors
13-15	-	reserved
16-19	STALINES	Physical number of lines printable (decimal) (from the generation or default values)
20-23	STACOLUM	Physical number of characters per line printable (decimal) (from generation or default values)
24	STATTYPE	Terminal type
	(STATYPE7) (STATYPE8)	Terminal can work in 7-bit mode only. Terminal can work in 7-bit or 8-bit mode.
25-32	STACURCH	Extended standard name. The value is returned only if the data display terminal supports 8-bit mode.
33	STACCSNN	Number of 8-bit character sets supported
	(STATRINF)	X' 00' and STATYPE8 simultaneously: the length of the information to be output exceeds the specified length. The information is truncated.
34	STACSS1	1st supported character set The variant number is specified in hexadecimal form.

Byte	Symb. name	Meaning
35	STACSS2	2nd supported character set The variant number is specified in hexadecimal form.
36	STACSS3	3rd supported character set The variant number is specified in hexadecimal form.
37	STACSS4	4th supported character set The variant number is specified in hexadecimal form.
38	STACSS5	5th supported character set The variant number is specified in hexadecimal form.
39	STACSS6	6th supported character set The variant number is specified in hexadecimal form.
40	STACSS7	7th supported character set The variant number is specified in hexadecimal form.
41	STACSS8	8th supported character set The variant number is specified in hexadecimal form.
42	STACSS9	9th supported character set The variant number is specified in hexadecimal form.
43	STACSS10	10th supported character set The variant number is specified in hexadecimal form.
44	STACSS11	11th supported character set The variant number is specified in hexadecimal form.
45	STACSS12	12th supported character set The variant number is specified in hexadecimal form.
46	STACSS13	13th supported character set The variant number is specified in hexadecimal form.
47	STACSS14	14th supported character set The variant number is specified in hexadecimal form.
48	STACSS15	15th supported character set The variant number is specified in hexadecimal form.
49	STACSS16	16th supported character set The variant number is specified in hexadecimal form.
50-51	-	Reserved
52-59	STAACTCH	Name of the extended character set activated. The name is returned only if the data display terminal supports 8-bit mode.
60	STARMODE	Physical read mode
	(STARMODM) (STARMODU)	Only modified fields are read. All unprotected fields are read.
61	STALLECH	Logical line-end symbol for data display terminals without equivalent hardware functions.

Byte	Symb. name	Meaning
62	STASUBCH	Substitute symbol for characters that are not X' 40' and are not logical control characters.
63	STAPERHC	Permanent hardcopy
	(STAPERHY)	All output messages for a data display terminal are printed out simultaneously via a conn. hardcopy unit.
	(STAPERHN)	Output messages are not also logged as hardcopy.

If less than 16 different variants are supported in STACCSn (bytes 34-49), the remaining variant numbers (16-n) are set to X'00'.

Examples see **TSTAT** macro.

DELFEI – Delete SOLSIG or POSSIG entry

General

Application area:	(Optimized) eventing; see page 94
Macro type:	Type R; see page 28

Forward eventing (FEV) is an optimized form of synchronous eventing. FEV avoids the need for repeated validation of the specified operands when **POSSIG** or **SOLSIG** calls are made repeatedly in a program.

Macro description

The macro **DELFEI** refers to an entry created in the EVENTLST using either **DSOFEI** or **DPOFEI** and deletes it from the EVENTLST. A **DISEI** call also deletes the associated forward events (implicit **DELFEI**).

Macro format and description of operands

DELFEI
REFNUM=(r)

REFNUM=
Identifies a register which (directly) contains the reference number for the POSSIG entry.

(r)
Register which contains the reference number.

Return information and error flags

During execution of the macro, register R1 contains the reference number; register R0 is overwritten with an internal function code.

R15:

b	b					a	a

A structured return code relating to the execution of the macro DELFEI is transferred in Register 15 (aa=primary return code, bb=secondary return code).

X'bb'	X'aa'	Meaning
X'00'	X'00'	Function executed: the entry in the EVENTLST has been deleted.
X'04'	X'04'	No action taken: incorrect reference number or entry already deleted.

DEQAR – Dequeue access request

General

Application area: (Task) serialization; see page 91
Macro type: Type S, MF format **1**: standard/L/E form; see page 29

DEQAR generates the 24-bit or the 31-bit interface, depending on the specification. In the event of macro chaining, the PARMOD operands of all macros chained must be given the same value.

Macro description

The **DEQAR** macro terminates access to the specified serialization item (previously requested with an **ENQAR** macro) by the task of the calling program. If another task is entered in the queue for this serialization item, it is activated and allowed access to the item. As an optional function, this macro can also be used to cancel the assignment (previously established with **ENASI**) of the serialization item to the caller task.
The CONTINU operand allows chaining of up to 255 **DEQAR** macros.

Macro format and description of operands

```
DEQAR
------------------------------------------------------------------------------------
⎧ ⎧ SINAME=name                              ⎫                                          ⎫
⎨ ⎨                                          ⎬,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL ⎬
⎪ ⎩ SINAMAD=addr / (r) [,SINAMLN=length]     ⎭                                          ⎪
⎩ SIID=addr / (r)                                                                       ⎭

,DISSI=NO / YES

,HOLDER=SELF / ANY

,CONTINU=NO / YES

[,PARMOD=24 / 31]

[,MF=L / (E, ..)]
```

SINAME=name
Specifies the name of the serialization item. The SCOPE operand must be added to ensure the serialization item is uniquely identified.

SINAMAD=
Specifies the address of the name of the serialization item. Identification is unique only if the SCOPE operand has also been specified.

addr
Symbolic address of the field containing the name.

(r)
Register containing the address.

SINAMLN=
Specifies the length in bytes of the serialization item name. The length must be at least 1 byte and not more than 54 bytes.
If the operand is missing, the length attribute of the SINAMAD operand is assumed if SINAMAD=addr is specified; if SINAMAD=(r), the maximum length (54) is assumed.

length
Length of the serialization item name.

SCOPE=
Specifies the scope of the serialization item (i.e. the participant tasks authorized to use it):

LOCAL
The serialization item is only used by the task of the calling program.

GROUP
All the tasks with the same user ID as the calling task.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.
The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system.

This is why the GETUGR macro (see the “SECOS” manual [14] has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

SIID=
Specifies the ID of the serialization item. The ID is provided to the user by the **ENASI** macro. If this ID is used instead of the name of the serialization item, processing time is reduced. A serialization item is uniquely identified by the ID.

addr
Symbolic address of a 4-byte field containing the ID.

(r)
Register containing the address.

DISSI=
Specifies whether use of the specified serialization item by the task of the calling program is to be terminated (see the **DISSI** macro).

NO
Use is not to be terminated.

YES
Use is to be terminated.

HOLDER=
Specifies whether access is to be terminated if the task of the calling program also requested the current access.

SELF
Access is terminated if the task of the calling program also requested the current access.

ANY
When HOLDER=ANY is specified, access is terminated irrespective of which task requested access.

CONTINU=
This operand allows chaining of up to 255 **DEQAR** macros.

NO
This is the last (or only) macro of a sequence.

YES
YES indicates that another **DEQAR** macro follows this macro.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Notes on the macro call

- When the list form (operand MF=L) of the macro is used, the following should be noted: Only one macro with MF=E need be specified for execution, regardless of whether this macro applies to a single request or to a series of requests.
- In the case of a series of requests, the operand list is generated via macro chaining (MF=L) by means of the CONTINU operand.
- When the program is terminated, all access requests of this program are terminated as well.

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the DEQAR macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'00'	X'00'	All dequeue macros were executed. No additional actions were taken.
X'04'	X'00'	All dequeue macros were executed. At least one serialization item was deleted.
X'08'	X'00'	All dequeue macros were executed. At least one disable operation (see the DISSI macro) was executed.
X'0C'	X'04'	Not all dequeue macros were executed. At least one dequeue operation was not executed because serialization items were not allocated to the task of the calling program.
X'10'	X'04'	Not all dequeue macros were executed. Invalid operands were specified: – Invalid address, e.g. address within a DSECT – Invalid length – Invalid name – SCOPE, CONTINU, DISABLE or HOLDER value undefined
X'14'	X'04'	Not all dequeue macros were executed. An invalid ID was specified (the ID is not known to the system, or the calling task did not issue an enable macro (see the ENASI macro)).
X'20'	X'04'	Not all dequeue macros were executed. No further access request existed for at least one serialization item, or access to at least one serialization item is to be terminated by means of HOLDER=SELF, but the access request was issued by another task. When several DEQAR macros are chained, the dequeue operations are performed for those serialization items for which they are permitted. The status of the serialization items for which theDEQUAR macro is invalid is not modified. If the DISSI=YES operand is specified, a disable function is performed.

DISCO – Disable contingency definition

General

Application area:	Contingency processing; see page 110
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

Macro description

The **DISCO** macro is provided to stop a routine from being used as a contingency process. The contingency process definition is disabled and therefore cannot be specified in subsequent **POSSIG** or **SOLSIG** macros.

Any subsequent **SOLSIG** and **POSSIG** macros which refer to the disabled contingency definition are rejected (return information). Any **SOLSIG** and **POSSIG** requests already queued (i.e. macros which have already been issued) are not affected when the contingency definition is disabled; in other words, the contingency process is initiated when the appropriate events occur.

Macro format and description of operands

```
DISCO
-------------------------------------------------------------
 ⎧ ⎧ CONAME=name                               ⎫ ⎫
 ⎪ ⎩ CONAMAD=addr / (r) [,CONAMLN=length]      ⎭ ⎪
 ⎨                                               ⎬
 ⎪ COID=addr / (r)                               ⎪
 ⎩                                               ⎭

[,PARMOD=24 / 31]
[,MF=L / (E,..)]
```

CONAME=name
Specifies the name of the contingency process.

CONAMAD=
Specifies the address of the name of the contingency process.

- **addr**
  Symbolic address of the field containing the name.

- **(r)**
  Register containing the address.

CONAMLN=
Specifies the length in bytes of the contingency process name.
The length must be at least 1 byte and not more than 54 bytes.
If the operand is missing, the length attribute of the CONAMAD operand is assumed if CONAMAD=addr is specified; if CONAMAD=(r), the maximum length (54) is assumed.

length
Length of the contingency process name.

COID=
Specifies the ID of the contingency process. The ID is supplied to the user by means of the **ENACO** macro.

addr
Symbolic address of a 4-byte field containing the ID.

(r)
Register containing the address.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Return information and error flags

Register R1 contains the operand list address.

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the DISCO macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'04'	X'00'	Contingency definition disabled.
X'10'	X'04'	Invalid operands specified. No action taken.
X'14'	X'04'	Invalid name or ID. No contingency process with the specified ID exists. No action taken.

For **examples**, see the section “Contingency processes” (page 110) and the **POSSIG** macro description (page 736).

DISEI – Disable event item

General

Application area: Eventing; see page 94

Macro type: Type S, MF format **1**: standard/L/E form;
see page 29

Macro description

The **DISEI** macro is provided to terminate the use of a specified event item by the task of the calling program. The event item is deleted if it is not used by any other task.

Macro format and description of operands

```
DISEI
--------------------------------------------------------------------------------------------------
⎧ ⎧ EINAME=name                                   ⎫                                                ⎫
⎪ ⎨                                               ⎬,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL    ⎪
⎨ ⎩ EINAMAD=addr / (r) [,EINAMLN=length]          ⎭                                                ⎬
⎪                                                                                                  ⎪
⎩ EIID=addr / (r)                                                                                  ⎭

[,PARMOD=24 / 31]
[,MF=L / (E,..)]
```

EINAME=name
Specifies the name of the event item that is no longer used by the calling task. The event item is uniquely identified only if the SCOPE operand is also specified.

EINAMAD=
Specifies the address of the name of the event item. For a unique event item definition, the SCOPE operand must also be specified.

addr
Symbolic address of the field containing the name of the event item.

(r)
Register containing the address.

EINAMLN=
Specifies the length of the event item name in bytes. The length must be at least 1 byte and not more than 54 bytes.
If the operand is missing, the length attribute of the EINAMAD operand is assumed if EINAMAD=addr is specified; if EINAMAD=(r), the maximum length (54) is assumed.

length
Length of the event item name.

SCOPE=
Specifies the scope of the event item (i.e. the participants authorized to use it).

LOCAL
Use of the event item is limited to the calling task.

GROUP
All the tasks with the same user ID as the calling task.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.

The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

EIID=
Specifies the event item ID. The ID is supplied to the user by the **ENAEI** macro. If this ID is used instead of the name of the event item, processing is speeded up. The ID is unique.

addr
Symbolic address of the field containing the ID.

(r)
Register containing the address of the field.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Functional description

If any **SOLSIG** or **POSSIG** requests by the calling program are still waiting in the queue of the event item when the **DISEI** macro is issued (see “Eventing”, page 94), these requests will be deleted; the assigned forward events will also be deleted from the EVENTLST (implicit **DELFEI**). If these were **SOLSIG** requests (synchronous/asynchronous), the calling program will be notified via the return code in register R15 or via the event information code. The same applies to **POSSIG** requests in which a contingency process was specified.

Return information and error flags

Register R1 contains the address of the operand list.

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the macro DISEI is transferred in register R15.

X'bb'	X'aa'	Meaning
X'04'	X'00'	Function executed: the event item was deleted.
X'08'	X'00'	Function executed: use of event item by the task of the calling program was terminated. The event item was not deleted.
X'0C'	X'04'	No action: the event item generated by the system is not assigned to the calling task.
X'10'	X'04'	No action: invalid operands were specified.
X'14'	X'04'	No action: invalid name or ID. No event item with the specified ID exists.
X'18'	X'04'	No action: the event item to be deleted is still being used by FASTPAM.

For **examples**, see the sections “Eventing” (page 106) and “Contingency processes” (page 118).

DISMP – Disable memory pool

General

Application area: Memory pools; see page 55
Macro type: Type S, MF format **1**: standard/L/E form; see page 29

A memory pool is a memory area (class 6 memory) that can be used by several users together. With **ENAMP**, a user can create a memory pool or declare participation in an existing memory pool. With **DISMP**, a user can explicitly terminate participation in a memory pool; participation is terminated implicitly with program termination.
A memory pool is addressed either via the pool name **or** via its ID (see **ENAMP**).
Following **DISMP**, participation in the same (still existing) memory pool must be declared again via **ENAMP**. The caller is given a new ID for the memory pool.

Macro description

A memory pool participant can sever the connection to the memory pool with the **DISMP** macro. The memory pool is deleted if the caller is the <u>last</u> (or only) user. All pages of the pool are implicitly released in this case.

Macro format and description of operands

```
DISMP
-----------------------------------------------------------------------------
 ⎧ ⎧                                             ⎫          ⎧ LOCAL      ⎫ ⎫
 ⎪ ⎪ MPNAME=name                                 ⎪          ⎪            ⎪ ⎪
 ⎪ ⎨          ⎧ addr ⎫            ⎧ length ⎫     ⎬,SCOPE=   ⎨ GROUP      ⎬ ⎪
 ⎨ ⎪ MPNAMAD= ⎨      ⎬[,MPNAMLN=  ⎨        ⎬]    ⎪          ⎪ USER_GROUP ⎪ ⎬
 ⎪ ⎩          ⎩ (r)  ⎭            ⎩ (r)    ⎭     ⎭          ⎩ GLOBAL     ⎭ ⎪
 ⎪                                                                         ⎪
 ⎪ MPID= ⎧ addr ⎫                                                          ⎪
 ⎩       ⎩ (r)  ⎭                                                          ⎭
[,PARMOD=24 / 31]
[,MF=L / (E,..)]
```

MPNAME=
Specifies the name of the memory pool. Note the connection with the SCOPE operand.

name
Name of the memory pool.

MPNAMAD=
Defines the address of the field with “name”.
(Note the connection with the SCOPE operand).

addr
Symbolic address (name) of the field.

(r)
Register containing the address value of the field.

MPNAMLN=
Defines the length of the name specified under MPNAMAD.
If not specified: length attribute of the addr field or, if MPNAMAD=(r) was specified, 54 bytes.

length
Length in bytes.

(r)
Register containing the length.

SCOPE=
Defines the scope (authorized users) of the memory pool. The specification is used to identify the memory pool uniquely and must always be specified in conjunction with the MPNAME and MPNAMAD operands.

LOCAL
The memory pool is used only by the user who created it.

GROUP
Memory pool users can be all the tasks with the same user ID as the user that created the memory pool.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.

The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

MPID=
Defines the address of a field (length = 4 bytes) with the ID for the memory pool (see **ENAMP**). The ID identifies the memory pool uniquely. Identifying the memory pool by the ID instead of by the name reduces processing time.

addr
Symbolic address (name) of the field containing the ID.

(r)
Register containing the address value of the field.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Return information and error flags

After macro processing, register R1 contains the operand list address.

R15:

b	b	0	0	0	0	a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the DISMP macro is transferred in register R15.
aa=X'00' : normal execution;
aa=X'04' : function not executed.

X'bb'	X'aa'	Meaning
X'0C'	X'00'	Normal execution; caller is the last or only participant; memory pool is deleted.
X'10'	X'00'	Normal execution; end of participation in memory pool; memory pool is retained.
X'04'	X'04'	Function was not carried out. The caller is not a participant in the memory pool in question (no ENAMP macro).
X'1C'	X'04'	Function was not carried out; operand error: – invalid address of the operand list – error in operand list structure – invalid address for MPNAMAD or MPID in the operand list – designation of the memory pool: – name contains invalid characters – invalid length specification (MPNAMLN) – memory pool not designated (MPNAME, MPNAMAD, MPID not specified) – MPNAMLN specified, but MPNAMAD omitted – SCOPE specified, but MPNAME/MPNAMAD omitted – designation not unequivocal: more than one operand was specified as designation (MPNAME/MPNAMAD/MPID) – invalid SCOPE specification – SCOPE=USER_GROUP was specified, although SRPM is not available in the system. – invalid register (R1) specification – PARMOD=24 specified in conjunction with 31-bit addressing mode (AMODE31)
X'24'	X'04'	Function not carried out; authorization error: – pages of the memory pool are still locked against releasing by the (user' s own) task; the lock was set in a privileged state – the caller is not authorized to terminate participation in a privileged or class 5 memory pool. – The access key at the time of the DISMP call does not match the access key valid when the memory pool was established. – The memory pool is still being used by FASTPAM.

31-bit interface:

– In the event of errors in the alignment or initialization of the standard header, the return codes X'0001FFFF' / X'0003FFFF'/ X'0004FFFF' are additionally transferred in register R15; see the table “Standard return codes” on page 43.

– No return codes are transferred in the standard header.

DISSI – Disable serialization item

General

Application area: (Task) serialization; see page 91
Macro type: Type S, MF format **1**: standard/L/E form; see page 29

DISSI generates a 24-bit or a 31-bit interface, depending on the the specification. In the event of macro chaining, all macros chained must make use of the same interface (either 24-bit or 31-bit interface).

Macro description

This macro is used to terminate utilization of the specified serialization item by the task of the calling program. If the serialization item is not used by any other task, it is deleted. Before the **DISSI** macro is called, the use of the serialization item must be disabled again (using the **DEQAR** macro).
The CONTINU operand allows up to 255 **DISSI** macros to be chained.

Macro format and description of operands

```
DISSI
--------------------------------------------------------------------------------------------
 ⎧ ⎧ SINAME=name                               ⎫                                                ⎫
 ⎨ ⎩ SINAMAD=addr / (r) [,SINAMLN=length]      ⎭,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL      ⎬
 ⎩   SIID=addr / (r)                                                                            ⎭

,CONTINU=NO / YES

[,PARMOD=24 / 31]

[,MF=L / (E,..)]
```

SINAME=name
Specifies the name of the serialization item. This specification is unique only in conjunction with the SCOPE operand.

SINAMAD=
Specifies the address of the name of the serialization item. This specification is unique only if the SCOPE operand is also specified.

addr
Symbolic address of the field containing the name.

(r)
Register containing the address.

SINAMLN=
Specifies the length in bytes of the serialization item name. The length must be at least 1 byte and not more than 54 bytes.
If the operand is missing, the length attribute of the SINAMAD operand is assumed if SINAMAD=addr is specified;
if SINAMAD=(r), the maximum length (54) is assumed.

length
Length of the serialization item name.

SCOPE=
Specifies the scope of the serialization item (i.e. the participants authorized to use it):

LOCAL
Use of the serialization item is limited to the calling task.

GROUP
All the tasks with the same user ID as the calling task.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.

The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [12]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

SIID=
Specifies the ID of the serialization item. This ID is supplied to the user by the **ENASI** macro. If this ID is used instead of the name of a serialization item, processing time is reduced. The ID is unique.

addr
Symbolic address of a 4-byte field containing the ID.

(r)
Register containing the address.

CONTINU=
Allows chaining of up to 255 **DISSI** macros.

<u>**NO**</u>
This is the last (or only) macro in a sequence.

YES
This **DISSI** macro is followed by another.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Notes on the macro call

- The disable function may also be requested as part of a **DEQAR** macro (see the **DEQAR** macro).
- The system performs a disable operation at program termination time for all serialization items still being used by the task.
- The following should be borne in mind when using the list form (operand MF=L) of the macro:
  Only one macro call with MF=E need be given for execution, irrespective of whether this call applies to a single request or to a series of requests. The operand list for a series of requests is generated by macro chaining (MF=L) with the aid of the CONTINU operand.

Return information and error flags

Register 1 contains the operand list address.

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the DISSI macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'04'	X'00'	All disable macros were executed: At least one serialization item was deleted.
X'08'	X'00'	All disable macros were executed: Use of all specified serialization items was terminated.
X'0C'	X'04'	Not all disable macros were executed: At least one serialization item was not used by the task of the calling program.
X'10'	X'04'	Not all disable macros were executed: Invalid operands were specified: – invalid address; e.g. address within a DSECT – invalid length – invalid name – SCOPE or CONTINU value undefined.
X'14'	X'04'	Not all disable macros were executed: An invalid ID was specified (the ID is not known to the system or the caller' s task did not issue an associated, valid enable macro (see the ENASI macro)).
X'24'	X'04'	Not all disable macros were executed: A disable request was issued for a serialization item for which an enqueue request was honored, but for which the associated dequeue function had not yet been performed.

DJINF – Create DSECT or data list for JINF macro

General

Application area: Requesting and accessing lists and tables; see page 155
Macro type: Definition macro; see page 28

Macro description

The **DJINF** macro generates a description of the output area for the **JINF** macro. The description is created in the form of a DSECT or data list. For the 24-bit interface, the DSECT/data list is generated by specifying the operand JATTR/RUNTIME; for the 31-bit interface, it is generated by specifying the PARLIST operand. In the latter case, the description of the output area starts with the standard header. The initialization values for the standard header are entered in the data list.

Macro format and description of operands

DJINF

DSECT=YES / NO
[,PREFIX=p]
{ ,JATTR=NO / YES ,RUNTIME=NO / YES
  ,PARLIST=NO / YES }

DSECT=
Specifies whether a DSECT of the output area is to be generated or a data list.

YES
A DSECT is generated.

NO
A data list is generated.

PREFIX=
Gives a character string to be prefixed to the symbolic names of the DSECT/data list.

p
Prefix for the symbolic names. Length ≤ 2 characters.
Default value: p = JI.

JATTR=
Indicates whether the DSECT/data list for job data is to be generated.

NO
No DSECT/data list is generated.

YES
A DSECT/data list is generated.

RUNTIME=
Indicates whether the DSECT/data list for current job data (start-of-job time, number of job retries) is to be generated.

NO
No DSECT/data list is generated.

YES
A DSECT/data list is generated.

PARLIST=
Indicates whether a DSECT/data list for the 31-bit interface is to be generated.

NO
No DSECT/data list is generated.

YES
A DSECT/data list is generated.

Layout of the DSECT for the 31-bit interface

```
           DJINF DSECT=YES,PARLIST=YES
1 *--------P A R A M E T E R L I S T
1          #INTF REFTYPE=REQUEST,                                        C
1                INTNAME=JINF,                                           C
1                INTCOMP=002
1 JIJOBDPL  DSECT
1          FHDR  UNIT=43,FUNCT=2,VERS=1
2          DS    0A
2          DS    0XL8                GENERAL OPERAND LIST HEADER
2          DC    AL2(43)             FUNCTION UNIT NUMBER
2          DC    AL1(2)              FUNCTION NUMBER
2          DC    AL1(1)              FUNCTION INTERFACE VERSION NUMBER
2          DC    X'FFFFFFFF'          Returncode is virgin
1 JIJOUID   DS    CL8                    USER ID
1 JIJOACC   DS    CL8                    ACCOUNT NUMBER
1 JIJOJCLA  DS    CL8                    JOB CLASS
1 JIJOJNAM  DS    CL8                    JOB NAME
1 JIJOTSN   DS    CL4                    TASK SEQUENCE NUMBER
1 JIJOJPRI  DS    X                      JOB PRIORITY
```

```
1 JIJORP    DS    X                        REPEAT-TYPE
1 JIJORPNO  EQU   X'01'                    - NO
1 JIJORPAS  EQU   X'02'                    - AT-STREAM-STARTUP
1 JIJORPDL  EQU   X'03'                    - DAILY
1 JIJORPWK  EQU   X'04'                    - WEEKLY
1 JIJORPPD  EQU   X'05'                    - PERIOD
1 JIJORPIT  DS    H                        REPEAT-INTERVAL
1 JIJOST    DS    X                        START-TYPE
1 JIJOSTSO  EQU   X'01'                    - SOON
1 JIJOSTEA  EQU   X'02'                    - EARLIEST
1 JIJOSTAT  EQU   X'03'                    - AT
1 JIJOSTLA  EQU   X'04'                    - LATEST
1 JIJOSTWI  EQU   X'05'                    - WITHIN
1 JIJOSTBU  EQU   X'06'                    - BYUSER
1 JIJOSTBO  EQU   X'07'                    - BYOPERATOR
1 JIJOSTIM  EQU   X'08'                    - IMMEDIATE
1 JIJOSTAS  EQU   X'09'                    - AT-STREAM-STARTUP
1 JIJOSTDR  DS    CL6                      START DATE REQUESTED
1 JIJOSTTR  DS    CL4                      START TIME REQUESTED
1 JIJORER   DS    X                        RERUN INDICATOR
1 JIJORERN  EQU   X'00'                    - NO
1 JIJORERY  EQU   X'80'                    - YES
1 JIJOFLU   DS    X                        FLUSH INDICATOR
1 JIJOFLUN  EQU   X'00'                    - NO
1 JIJOFLUY  EQU   X'80'                    - YES
1 JIJOTIME  DS    F                        CPU TIME REQUESTED
1 JIJONTL   DS    X                        NO TIME LIMIT INDICATOR
1 JIJONTLN  EQU   X'00'                    - NTL NOT REQUESTED
1 JIJONTLY  EQU   X'80'                    - NTL REQUESTED
1 JIJOMONJ  DS    CL54                     MONITORING JOB VARIABLE
1 JIJOSTDA  DS    CL6                      DATE JOB ACTUALLY STARTED
1 JIJOSTTI  DS    CL4                      TIME JOB ACTUALLY STARTED
1 JIJORPCO  DS    H                        REPEAT-COUNT
1 JIJOLEN   EQU   *-JIJOBDPL               LENGTH OF PARAMETERLIST
```

DJSI – Create DSECTs or data areas for job scheduler macros (24-bit interface)

General

Application area:	Job scheduler (system administration macro); see page 165
Macro type:	Definition macro; see page 28

- JMS = Job Management System; JSS = Job Scheduling Supports.
  JSS is part of JMS.
- For 31-bit interface, see the **DJSIPL** macro.

Macro description

The **DJSI** macro creates name definitions, DSECTs and data areas for the 24-bit interface of the following job scheduler macros:

JSATTCH	Attach job scheduler to Job Management System
JSDETCH	Detach job scheduler from Job Management System
JSEXPCT	Request next event for job scheduler
JSRUNJB	Transfer job to class scheduler
JSINFO	Transfer STREAM-PARAMETER values
JSWAKE	Specify next time event for job scheduler

DSECTs and data lists start with the standard header. The initialization values for the standard header are entered.

Macro format and description of operands

DJSI
DSECT=YES / NO [,PREFIX=p] ,EVENT=NO / YES ,CLOCK=NO / YES ,JSINF=NO / YES ,STRTINF=NO / YES ,WAKE=NO / YES

DSECT=
Specifies whether a dummy section (DSECT) is to be generated for the subsequently specified data areas or whether data areas and definitions are created directly in the application program.

YES
A dummy section is generated.

NO
The data areas and definitions are created and stored directly in the application program (immediate access via the symbolic names).

EVENT=...
Specifies whether the output area of the **JSEXPT** macro and a list of JMS events are to be created.

CLOCK=...
Specifies whether the output area of the **JSATTCH** macro is to be created.

STRTINF=...
Specifies whether the input area of the **JSRUNJB** macro is to be created.

JSINF=...
Specifies whether the output area of the **JSINFO** macro is to be created.

WAKE=...
Specifies whether the input area of the **JSWAKE** macro is to be created.

PREFIX=
Gives a character string to be prefixed to the symbolic names of the DSECT / I/O area.

p
Prefix for the symbolic names. Length ≤ 2 characters. Default value: p = JS.

Notes on the macro call

Equates for the return codes of the job scheduler macros are appended to every subset (EVENT, CLOCK,...).

DJSIPL – Create DSECTs or data areas for job scheduler macros (31-bit interface)

General

Application area: Job scheduler (system administration macro); see page 165
Macro type: Definition macro; see page 28

- For 24-bit interface please refer to the **DJSI** macro.

Macro description

The **DJSIPL** macro creates name definitions, DSECTs and data areas for the 31-bit interface of the following job scheduler macros:

JSATTCH	Attach job scheduler to Job Management System
JSDETCH	Detach job scheduler from Job Management System
JSEXPCT	Request next event for job scheduler
JSRUNJB	Transfer job to class scheduler
JSINFO	Transfer STREAM-PARAMETER values
JSWAKE	Specify next time event for job scheduler

DSECTs and data areas start with the standard header. The initialization values for the standard header are entered.

Macro format and description of operands

DJSIPL
DSECT=YES / NO [,PREFIX=p] ,JSATTCH=NO / YES ,JSEXPCT=NO / YES ,JSINFO=NO / YES ,JSDETCH=NO / YES ,JSRUNJB=NO / YES ,JSWAKE=NO / YES

DSECT=
Specifies whether a dummy section (DSECT) is to be generated for the subsequently specified data areas or whether data areas and definitions are created directly in the application program.

YES
A dummy section is generated.

NO
The data areas and definitions are created and stored directly in the application program.

JSATTCH=...
Specifies whether the output area of the **JSATTCH** macro is to be created.

JSDETCH=...
Specifies whether the input area of the **JSDETCH** macro is to be created.

JSEXPCT=...
Specifies whether the output area of the **JSEXPCT** macro and a list of JMS events are to be created.

JSRUNJB=...
Specifies whether the input area of the **JSRUNJB** macro is to be created.

JSINFO=...
Specifies whether the output area of the **JSINFO** macro is to be created.

JSWAKE=...
Specifies whether the input area of the **JSWAKE** macro is to be created.

PREFIX=
Gives a character string to be prefixed to the symbolic names of the DSECTs or data areas.

p
Prefix for the symbolic names. Length ≤ 2 characters. Default value: p = JI.

DPOFEI – Create POSSIG entry

General

Application area:	(Optimized) eventing; see page 94
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

Forward eventing (FEV) is an optimized form of synchronous eventing. FEV avoids the necessity for repeated validation of the specified operands when **POSSIG** or **SOLSIG** calls are made repeatedly in a program. Instead, an event list (EVENTLST) is created and if, for example, signals are to be sent to an event item (using the **POSSIG** function), a single entry is made in the list. In subsequent stages of the program when (real) send requests are issued, they will simply reference this entry each time (using **RPOFEI**). The entry may be explicitly deleted again (using **DELFEI**).
A maximum of 2047 entries may be generated per participant in the EVENTLST. The task of the calling program must be assigned to the event item (using **ENAEI**).

Macro description

The macro **DPOFEI** creates a POSSIG entry in the event list EVENTLST. All the necessary details are copied into the entry (name of the event item or event item ID, post code, maximum time for collecting the signal (event)). A reference number for the entry is passed back to the caller.
It is possible to chain together several consecutive calls to **DPOFEI**. The chain can also be terminated using the macro **DSOFEI**. When using a chain, note that:

– **DSOFEI** must always appear as the last macro in the chain.

– The first macro call which results in an error will terminate the chain; any entries already created will be deleted.

– The operand REFNUM (for communicating the reference number) may only be specified in the first macro call in the chain.

– A maximum of 5 macro calls may be chained together. All macros in the chain must use the same interface format.

Macro format and description of operands

```
DPOFEI
-------------------------------------------------------------------------------------------------
 ⎧ ⎧ EINAME=name                             ⎫                                                  ⎫
 ⎨ ⎩ EINAMAD=addr / (r) [,EINAMLN=length]    ⎭,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL        ⎬
 ⎩  EIID=addr / (r)                                                                             ⎭

[,REFNUM=addr / (r)]

[, ⎧ SPOSTAD=addr / (r) ⎫ ]
   ⎩ SPOSTR=r           ⎭

,SPOSTL=1 / 2
,CONTINU=NO / YES / DSOFEI
[,LIFETIM=sec / (r)]
[,PARMOD=24 / 31]
[,MF=L / (E,..)]
```

EINAME=
Indicates that the name which follows is that which the caller assigned to the event item, using the macro **ENAEI** (note the interaction with the SCOPE operand).

name
Name of the event item.

EINAMAD=
Indicates that the address which follows is that of a field which contains the name of the event item (note the interaction with the SCOPE operand).

addr
Symbolic address of the field containing the name of the event item.

(r)
Register containing the address value “addr”.

EINAMLN=
Indicates that the length of the name of the event item follows.
If this operand is omitted, the length of the name is taken to be the same as for the field specified in EINAMAD=addr, or 54 bytes if EINAMAD=(r) was used.

length
Length of the name in bytes.

SCOPE=
Specifies the scope of the event item (i.e. participants authorized to use it). The name of the event item is unique only in combination with the scope. SCOPE must always be specified in conjunction with EINAME or EINAMAD.

LOCAL
Use of the event item is limited to the caller's task.

GROUP
All the tasks with the same user ID as the caller's task are participants.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.

The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

EIID=
Specifies the location of the event item ID.
This event item ID is passed to the user when the macro **ENAEI** is executed. It provides a unique identifier for the event item; its use results in faster execution of the macro.

addr
Symbolic address of a 4-byte field containing the event item ID.

(r)
Register containing the address value “addr”.

REFNUM=
Indicates that the address which follows identifies a field in which the reference number for the EVENTLST entry is passed to the caller.
Field length = 4 bytes; aligned on a word boundary.

addr
Symbolic address of the field in which the reference number is to be stored.

(r)
Register containing the address value “addr”.

SPOSTAD=
Indicates that the address which follows is for a field containing a post code which is to be sent to the receiver of the signal (see the macro **SOLSIG** or **DSOFEI**).
The post code has a length of 4 or 8 bytes. Field length = 4 or 8 bytes; aligned on a word boundary.
A post code in the form X'00000000' will not be sent. The operand SPOSTR permits the same function to be executed more rapidly.

addr
Symbolic address of the field containing the post code.

(r)
Register containing the address value “addr”.

SPOSTR=
Indicates a register which (directly) contains the post code. If the 8-byte post code (=2 words) is used, the register following the specified register (in number) must contain the second word of the post code. A post code in the form X'00000000' will not be sent.

r
Register containing the post code.

SPOSTL=
Gives the post code length in words.

<u>1</u>
Post code length = 1 word (=4 bytes).

2
Post code length = 2 words.

CONTINU=
Enables the **DPOFEI** macro to be chained with other, immediately succeeding **DPOFEI** or **DSOFEI** calls.

<u>NO</u>
No further **DPOFEI** or **DSOFEI** call follows.

YES
A **DPOFEI** macro call follows.

DSOFEI
A **DSOFEI** macro call follows.

LIFETIM=
Can be used to specify a time interval (in seconds), within which the signaled event must be collected (using the macro **SOLSIG** or **DSOFEI**). After this time interval has expired, the event will be deleted from the event queue.
Execution may be delayed by up to +10 seconds.

sec
Time specification in seconds.
1 ≤ seconds ≤ 43200. Default value: seconds = 600.

(r)
Register containing the value of “sec”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Return information and error flags

During execution of the macro, register R1 contains the address of the operand list; register R0 is overwritten.

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the DPOFEI macro is transferred in R15.

X'bb'	X'aa'	Meaning
X'00'	X'00'	Normal execution. A POSSIG entry was created. All the specified operands are valid.
X'04'	X'04'	No action taken: the permitted maximum of 2047 calls has been exceeded.
X'0C'	X'04'	No action taken: the event item is not enabled for the task of the caller program.
X'10'	X'04'	No action taken: invalid operand specified.
X'14'	X'04'	No action taken: there is no event item with the specified name or ID.

Example of a chained macro structure:

```
DPOFEI   START
         BALR
         USING
         . . .
         DPOFEI EIID=IDP1,REFNUM=REFNR,CONTINU=YES
         DPOFEI EIID=IDP2,CONTINU=YES
         DPOFEI EIID=IDP3,CONTINU=DSOFEI
         DSOFEI EIID=IDS1
         . . .
         TERM
*****  DEFINITIONS  ******************************
         . . .
IDP1     DS    CL4
IDP2     DS    CL4
IDP3     DS    CL4
IDS1     DS    CL4
REFNR    DS    F
         . . .
         END
```

DSHARE – Unload user's shared code from common memory pool

General

Application area:	Linking and loading; see page 47
Macro type:	Type S, MF format **2**: standard/C/D/L/E/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **DSHARE** macro unloads a single shareable program from a common memory pool. This program was loaded previously by means of the **ASHARE** macro. The calling task must be connected to the common memory pool containing the shareable program involved. When the last participant is disconnected from a memory pool, all the shareable programs in this memory pool are unloaded and the memory pool is deleted (see the **DISMP** macro).

Macro format and description of operands

```
DSHARE

  { PROGRAM=name      }
  { PROG@=addr / (r)  }

[, { PGMVERS=*STD / version }  ]
   { PGMVER@=addr / (r)     }

,MF=S / C / D / E / L / M
[,PARAM=addr / (r)]
,PREFIX=P / p
,MACID=BDS / macid
```

PROGRAM=name
Identifies the program that was loaded into the common memory pool using the **ASHARE** macro. The specified name must be unique and may be up to 32 characters long.

PROG@=
Specifies the address of a field containing the program name. May be specified only in conjunction with MF=M.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

PGMVERS=
Specifies the program version.

***STD**
The program version is not taken into account during unloading, i.e. the first program found with the specified name is unloaded.

version
The version specified may be up to 24 characters long. If this program version does not exist in the common memory pool, nothing is unloaded and the corresponding return code is output.

PGMVER@=
Specifies the address of a field containing the program version. May be specified only if MF=M.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM, PREFIX and MACID, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form, D form or M form of the macro, and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

Notes on the macro call

- Before calling **DSHARE**, the user must be connected to the memory pool.
- Only the **DSHARE** macro can be used to unload shared code from memory pools. The **UNBIND** macro may not be used for this purpose.
- It is the user's responsibility to check whether the program to be unloaded is still being executed by other users.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the DSHARE macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally.
X'00'	X'01'	X'0001'	The program name is missing.
X'00'	X'01'	X'0010'	The program was not found in any memory pool accessible to the user.
X'00'	X'01'	X'0011'	The program was found but the user is not connected to the memory pool involved.
X'00'	X'20'	X'0100'	System error.
X'00'	X'20'	X'0101'	DBL error during unloading.
X'00'	X'20'	X'0103'	DBL-LOCK-MANAGER error during the DSHARE macro processing
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported.
X'00'	X'03'	X'FFFF'	The interface version is not supported.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

DSOFEI – Create SOLSIG entry

General

Application area:	(Optimized) eventing; see page 94
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

Forward Eventing (FEV) is an optimized form of synchronous eventing. FEV avoids the necessity for repeated validation of the specified operands when **SOLSIG** or **POSSIG** calls are issued repeatedly in a program. Instead, an event list (EVENTLST) is created and if, for example, signals are to be requested from an event item (using the **SOLSIG** function), a single SOLSIG entry is made in the list. In subsequent stages of the program, whenever a (real) receive request is issued (using **RSOFEI**) it will simply refer to the EVENTLST entry. The entry can be explicitly deleted again (using **DELFEI**).

A maximum of 2047 entries may be generated per participant in the EVENTLST. The task of the calling program must be assigned to the event item (by a preceding **ENAEI**).

Macro description

The macro **DSOFEI** creates a SOLSIG entry in the event list EVENTLST. All the necessary details are copied into the entry (name of the event item or ID, post code, maximum waiting time allowed for the signal (event) to occur. A reference number for the entry is passed to the caller.

Macro format and description of operands

```
DSOFEI
--------------------------------------------------------------------------------
 ⎧ ⎧ EINAME=name                              ⎫                                               ⎫
 ⎨ ⎩ EINAMAD=addr / (r) [,EINAMLN=length]     ⎭,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL     ⎬
 ⎩ EIID=addr / (r)                                                                            ⎭

,REFNUM=addr / (r)

[,LIFETIM=sec / (r)]

[, ⎧ RPOSTAD=addr / (r) ⎫], RPOSTL=1 / 2, RPOSTNUM=number / (r)
   ⎩ RPOSTR=r           ⎭

[,PARMOD=24 / 31]

[,MF=L / (E,..)]
```

EINAME=
Indicates that the name which follows is that which the caller assigned to the event item, using the macro **ENAEI** (note the interaction with the SCOPE operand).

name
Name of the event item.

EINAMAD=
Indicates that the address which follows is that of a field which contains the name of the event item (note the interaction with the SCOPE operand).

addr
Symbolic address of the field containing the name of the event item.

(r)
Register containing the address value “addr”.

EINAMLN=
Indicates that the length of the name of the event item follows.
If this operand is omitted, the length of the name is taken to be the same as for the field specified in EINAMAD=addr, or 54 bytes if EINAMAD=(r) was used.

length
Length of the name in bytes.

SCOPE=
Specifies the scope of the event item (i.e. participants authorized to use it). The name of the event item is unique only in combination with the scope. SCOPE must always be specified in conjunction with EINAME or EINAMAD.

LOCAL
Use of the event item is limited to the caller's task.

GROUP
All the tasks with the same user ID as the caller's task are participants.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.

The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

EIID=
Specifies the location of the event item ID.
This event item ID is passed to the user when the macro **ENAEI** is executed. It provides a unique identifier for the event item; its use results in faster execution of the macro.

addr
Symbolic address of a 4-byte field containing the event item ID.

(r)
Register containing the address value “addr”.

REFNUM=
Indicates that the address which follows identifies a field in which the caller is to receive the reference number for the entry in the EVENTLST.
Field length = 4 bytes; aligned on a word boundary.

addr
Symbolic address of the field into which the reference number is to be put.

(r)
Register containing the address value “addr”.

LIFETIM=
Can be used to specify a time interval (in seconds), which is the maximum time that the task of the calling program should wait for the occurrence of the event (signal). Termination of the wait may be delayed by up to +10 seconds.

sec
Time specification in seconds. $1 \le \text{sec} \le 43200$.
Default value: sec = 600.

(r)
Register containing the value of “sec”.

RPOSTAD=
Indicates that the address which follows is for a field in which a post code may be entered (see the macro **POSSIG**).
The post code has a length of 4 or 8 bytes. The RPOSTL operand determines whether both words of the post code or just the 1st word are to be transferred to this field.
The operand RPOSTR permits the same function as RPOSTAD to be executed more rapidly.

addr
Symbolic address of the field in which the post code is to be entered. Field length = 4 or 8 bytes.

(r)
Register containing the address value “addr”.

RPOSTR=
Specifies a register in which the post code it to be entered directly. A post code consisting of 2 words is entered in the specified register and the one following it (in number) if RPOSTL=2 is specified.

r
Register to receive the post code.

RPOSTL=
Gives the number of words to be received as post code.

1
Only one word (the first) of the post code is to be transferred.

2
The complete post code (= 2 words) is to be transferred.

RPOSTNUM=
Specifies the maximum number of post codes which are to be transferred. The range specified for RPOSTAD must be large enough to be able to receive all post codes. Required length: `RPOSTNUM*2+1` (1 byte indicator X'FF', end of list).

number
Number of post codes to be transferred.

(r)
r = register with the number of post codes to be transferred.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Return information and error flags

During execution of the macro, register R1 contains the address of the operand list; register R0 is overwritten.

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the DSOFEI macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'00'	X'00'	Normal execution. A SOLSIG entry has been created. All the specified operands are valid.
X'04'	X'04'	No action taken: the permitted maximum of 2047 calls has been exceeded.
X'0C'	X'04'	No action taken: the event item is not enabled for the task of the calling program.
X'10'	X'04'	No action taken: invalid operand specified.
X'14'	X'04'	No action taken: there is no event item with the specified name or ID.

DSPSRV – Control a data space

General

Application area:	Extended addressing with data spaces; see page 61
Macro type:	Type S, MF format **3**: C/D/L/M/R/E form; see page 29

The **DSPSRV** macro can be used all BS2000 servers
(see section “Extended addressing with data spaces” on page 61).

Macro description

The **DSPSRV** macro enables the user to create a data space by specifying type, name, scope and desired size. A user who creates a data space automatically becomes its owner. The system returns an identification (the SPID) that uniquely identifies the data space throughout the session. A program that is to access a data space must be connected to it via access lists. This connection is established by means of the **ALESRV** macro.

The data space type determines the kind of memory allocation/deallocation within the data space and the functions that are provided for this.

The functions of the **DSPSRV** macro make it possible to:

- create a data space (FCT=CREATE),
- release an existing data space (FCT=DESTROY),
- request information on an existing data space (FCT=INFORM),

for a data space with type STACK:

- extend an existing data space by adding memory pages (FCT=EXTEND) and
- delete the contents of an existing data space (in some cases also in units of 4K), i.e. overwrite them with binary zeros (FCT=CLEAR)
- reduce the current size of a data space (FCT=REDUCE)

for a data space with type HEAP:

- allocate an area (FCT=GETAREA)
- release an allocated area (FCT=RETAREA)

Macro format and description of operands

```
DSPSRV

      ⎧CREATE, NAME='name'/name_addr, INISIZE=number / (r)                       ⎫
      ⎪     ,MAXSIZE=number / (r), DIAPROT=NO/YES,                                ⎪
      ⎪     ,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL                            ⎪
      ⎪     ,TYPE=STACK / HEAP                                                    ⎪
      ⎪DESTROY,SPID=spid_addr                                                     ⎪
      ⎪INFORM,IDENT=NAME / SPID [,SPID=spid_addr][,NAME='name' / name_addr]      ⎪
FCT=  ⎨     ,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL                            ⎬
      ⎪EXTEND,SPID=spid_addr ,SIZE=number / (r)                                   ⎪
      ⎪CLEAR,SPID=spid_addr,AREA=area_addr / (r) ,SIZE=number / (r)               ⎪
      ⎪REDUCE,SPID=spid_addr, SIZE=number / (r)                                   ⎪
      ⎪GETAREA,SPID=spid_addr, SIZE=number / (r)                                  ⎪
      ⎩RETAREA,SPID=spid_addr, AREA=area_addr / (r) ,SIZE=number / (r)            ⎭

[,MF=C / D / L / M / E / R[,SPID=spid_addr][,EXTADDR=ext_addr][.AREA=area_addr]]

[,PARAM=addr / (r)]

,PREFIX=N / p

,MACID=VDD / macid
```

The operands are described in alphabetical order below.

AREA=
Specifies the start address of a data space area. This operand can be both input operand (FCT=RETAREA, CLEAR) and output operand (FCT=GETAREA).
The address must be aligned on a page boundary (4K) and be located within the area of the generated data space.

area_addr
Symbolic address (name) of a 4-byte field containing the start address of the area.

(r)
Register containing the start address value “addr”.

DIAPROT=
Specifies whether the data space is to be protected against access by diagnostic tools (e.g. AID, USERDUMP, CDUMP2).

NO
The data space is not protected, i.e. diagnostic tools may access it.

YES
The data space is to be given special protection. Diagnostic tools may not access it.

EXTADDR=
Specifies the start address of the data space to be extended. This operand is an output operand (only with MF=R).

ext_addr
Address for outputting the start address of the new memory space extension (with FCT=EXTEND).

FCT=
Specifies which of the functions of the **DSPSRV** macro is to be executed.

CREATE
Creates a new data space. The caller becomes the owner of the data space.
With return code X'aaaa'=X'0000', the system returns the SPID. This can be read from the parameter list with MF=R.

DESTROY
Releases an existing data space, provided that the caller is also the owner of the data space.

EXTEND
Extends an existing data space of type STACK by adding 4K memory pages. For information on the size of the data space, see the notes on address space size.
With return code X'aaaa'=X'0000', the system returns the start address of the memory space extension (EXTADDR). This can be read from the parameter list with MF=R. The allocated area within the data space is cleared to binary zeros.

CLEAR
Deletes the contents of a data space area of type STACK by overwriting memory pages in units of 4K with binary zeros. The deleted memory pages are no longer retained in real memory (paging memory).

INFORM
Provides information on the data space specified by means of its name and scope or SPID. The DSECT generated with MF=D contains all information.

REDUCE
Reduces the current size of a data space of type STACK in 4 K units.

GETAREA
Allocates an area within a data space of type HEAP.
With return code X'aaaa'=X'0000', the system returns the start address of the area (AREA). This can be read from the parameter list with MF=R. The allocated area within the data space is cleared to binary zeros.

RETAREA
Releases an allocated area within a data space of type HEAP.

IDENT=
Specifies which operands (NAME and SCOPE or SPID) are to be used to identify the data space (if FCT=INFORM).

NAME
The data space is identified by means of its name and scope.

SPID
The data space is identified by means of its SPID.

INISIZE=
Specifies the initial size of the requested data space in units of 4K.
For information on the size of the data space, see the notes on address space size and allocation size on page 445.

number
Positive integer (X'01' .. X'80000') specifying the initial size of the data space.

(r)
Register containing the “number”.

MAXSIZE=
Specifies the maximum desired size of the requested data space in units of 4K.
For a data space of the type HEAP the specified size is rounded up to the next MB boundary.
A data space does not have to reach the maximum size specified here; this value simply sets an upper limit. The maximum permitted size depends on the maximum address space size (ADDRESS-SPACE-LIMIT) specified in the user catalog (see the notes on address space size).

number
Positive integer (X'01' .. X'80000') specifying the maximum size of the data space.

(r)
Register containing the “number”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form, D form, R form or M form of the macro and additionally a MACID in the C form, R form or M form (see page 29).

MF=R enables the output parameters of the functions CREATE (SPID=), INFORM (SPID=), EXTEND (EXTADDR=) and GETAREA (AREA=) to be read from the parameter area.

NAME=
Specifies the name of the data space. Length = 1..54 alphanumeric characters, the first of which must be a letter or one of the characters # or @.
The name of a data space is unique only within its scope (see the SCOPE operand), i.e. there may be data spaces with the same name but with different scopes.

'name'
Name of the data space.

name_addr
Symbolic address (name) of a field (54 byte) containing the name of the data space in alphanumeric characters.

SCOPE=
Defines the scope of the specified data space. The name of a data space identifies the data space uniquely only within the specified scope.
The scope determines which tasks can participate in the specified data space, i.e. access it.

LOCAL
Use of the data space is limited to the task which created it. Other tasks may not access it.

GROUP
All tasks with the user ID of the task that created the data space can be connected to the data space.

USER_GROUP
All tasks belonging to the same user group as the user ID of the task that created the data space can use the data space, provided that the SRPM has been loaded.

GLOBAL
All the tasks in the system can access the data space.

SIZE=
Specifies the size of the data space area for the functions EXTEND, REDUCE, CLEAR, GETAREA and RETAREA. This operand is specified in units of 4K, see notes on allocation size on page 446.
The following points must be taken into account:

- if FCT=EXTEND: The data space extended by SIZE may not exceed the size specified in MAXSIZE; see the notes on address space size on page 445.
- if FCT=CLEAR/RETAREA: The area defined by AREA and SIZE must be located within an existing data space identified by means of its SPID.
- if FCT=REDUCE: The value of SIZE may not be greater than the current size of the data space.
- if FCT=GETAREA: The sum of all allocated areas may not be greater than specified in MAXSIZE.

number
Positive integer ≥ 1 that defines the number of 4K units to be added to the data space (FCT=EXTEND) or deleted (FCT=CLEAR).

(r)
Register containing the address value of “number”.

SPID=
Identifies a data space uniquely throughout the system.
The system assigns the SPID when a data space is created. This operand may be an input or output operand.

spid_addr
Symbolic address (name) of an 8-byte field containing the SPID of the data space.

TYPE=
Determines the type of memory allocation/deallocation within the data space. This is determined at generation time.

STACK
A data space of the type STACK is an allocated area that is contiguous in virtual terms, starting with address 0 and going up to the current size.
The allocation functions available are EXTEND, REDUCE and CLEAR. The GETAREA and RETAREA functions are rejected.

HEAP
A data space of the type HEAP is a virtual address space in which areas of any required size can be allocated dynamically up to the maximum size of the data space. The allocation functions available are GETAREA and RETAREA. The EXTEND, REDUCE and CLEAR functions and the INISIZE parameter are rejected.

Note

DIV (Data in Virtual) functionality is only supported in data spaces of the type STACK.

Notes on the macro call

- The SPID may be an input or output operand, i.e. it is placed in the generated parameter list as an output operand with FCT=CREATE or FCT=INFORM and, the next time this parameter list is used, is also valid as an input operand. The same applies for the AREA operand in the functions GETAREA and RETAREA.
- The sum of the memory pages already occupied by the task and the additional memory pages requested by **DSPSRV** may not exceed the maximum address space size (ADDRESS-SPACE-LIMIT) entered in the user catalog for the owner. The same applies to the MAXSIZE specification.
  The command SHOW-USER-ATTRIBUTES PUBSET=*HOME can be used to obtain information on the user's own address space size. If the address space is full, a storage space error is reported and the function is aborted.
- On termination of the program that created the data space, the data space is released automatically, i.e. without FCT=DESTROY. The SPID is no longer valid; any remaining ALETS, however, are not deleted. If one of these ALETs is used to attempt to access a data space that no longer exists, the system issues an error message.
  If a data space to be deleted still contains DIV windows, these windows must be closed before the data space can be released.
  All these error situations are listed in the error flags (return codes, see below).

Notes on address space size

- Any specification for creating or extending data spaces may not exceed the maximum address space size specified in the user catalog. This means that the sum of all memory requests must be less than or equal to the address space size given in the ADDRESS-SPACE-LIMIT field. The sum of all memory requests is calculated from:
  - the class 6 memory pages allocated in the program space,
  - the allocated pages that were requested for existing data spaces,
  - the pages requested in the current **DSPSRV** macro (with INISIZE and MAXSIZE when creating a data space or with SIZE when extending a data space).
- The maximum size specified for a data space when it is created (MAXSIZE) must be greater than or equal to the initial data space size specified for INISIZE.
  The size specified in SIZE when extending a data space (together with INISIZE and any memory pages that were added previously with FCT=EXTEND) may not exceed MAXSIZE.

Notes on allocation size

– The size of an area (INISIZE=, SIZE=) is always specified in units of 4 KB in the CREATE, EXTEND, REDUCE, GETAREA and RETAREA functions.
However, the actual allocation/deallocation always takes place in units of the hardware page size (4 KB for all BS2000 servers). From the user viewpoint this only means that the accessible area can be greater than the area actually requested.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the DSPSRV macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function executed successfully. The following values can be read from the parameter list with MF=R: – if FCT=CREATE: spid_addr (SPID operand) – if FCT=EXTEND: ext_addr (EXTADDR operand) – if FCT=INFORM: spid_addr (SPID operand) – if FCT=GETAREA: area_addr (AREA operand)
X'02'	X'00'	X'0001'	Warning: the specified data space has been released even though other programs are connected to it (if FCT=DESTROY).
X'00'	X'01'	X'0003'	Invalid FCT operand.
X'01'	X'01'	X'0003'	Invalid NAME operand.
X'02'	X'01'	X'0003'	Invalid SCOPE operand.
X’04’	X’01’	X’0003’	Invalid TYPE operand
X'05'	X'01'	X'0003'	Invalid IDENT operand.
X'06'	X'01'	X'0003'	Invalid MAXSIZE operand.
X'07'	X'01'	X'0003'	Invalid INISIZE operand.
X'0A'	X'01'	X'0003'	Invalid DIAPROT operand.
X'0C'	X'01'	X'0003'	Invalid AREA operand.
X'0D'	X'01'	X'0003'	Invalid combination of operands.
	X'20'	X'0005'	Internal error.
X'00'	X'40'	X'000D'	DIV application running: the data space contains DIV windows (if FCT=DESTROY).
X'00'	X'40'	X'0102'	Invalid specification of NAME operand. A data space with the specified name already exists (only if FCT=CREATE).
X'00'	X'40'	X'0104'	Invalid specification of NAME and/or SCOPE (if FCT=INFORM).
X'00'	X'40'	X'0106'	Maximum load reached for paging memory.

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'40'	X'0107'	The maximum address space available to the task (ADDRESS-SPACE-LIMIT in user catalog) has been exceeded. Invalid specification of INISIZE or MAXSIZE (if FCT=CREATE) or SIZE (if FCT=EXTEND).
X'00'	X'40'	X'0202'	Error in SCOPE=USER_GROUP: the SRPM subsystem is not loaded (if FCT=CREATE).
X'00'	X'40'	X'0206'	Maximum load reached for main memory.
X'00'	X'40'	X'0302'	The calling task is not the owner of the data space and may not delete it (if FCT=DESTROY).
X'00'	X'40'	X'0304'	Incorrect specification of SPID operand. The specified data space does not exist or the caller is not authorized to access this data space.
X'00'	X'40'	X'0306'	Maximum number of data spaces reached (if FCT=CREATE).
X'00'	X'40'	X'0404'	Error in TYPE. The data space type is invalid for the specified function.
X'00'	X'40'	X'0406'	Address space saturation. There is insufficient free and contiguous address space available in the data space to satisfy the requirements (with FCT=GETAREA)
X'00'	X'40'	X'0604'	Invalid MAXSIZE operand: – maximum data space size exceeded (if FCT=EXTEND). – the specified size is greater than the current size of the data space (with FCT=REDUCE). – the specified size is greater than the maximum size of the data space (with FCT=GETAREA)
X'00'	X'40'	X'0C04'	The specified area is not a part of the data space (if FCT=CLEAR or RETAREA).
X'00'	X'40'	X'0F04'	Allocation error. The area specified is not allocated within the data space (with FCT=RETAREA).
X'00'	X'81'	X'0106'	Maximum load reached for paging memory.
X'00'	X'81'	X'0306'	Internal shortage of resources.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

For an **example** see section “Extended addressing with data spaces” on page 68.

DTMODE – Create DSECT or data list for TMODE macro

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Definition macro; see page 28

Macro description

The **DTMODE** macro generates a description of the input/output area for the **TMODE** macro in 31-bit addressing mode. The description is created in the form of a DSECT or a data list and starts with the standard header. The initialization values for the standard header are entered in the data list.

Macro format and description of operands

DTMODE
DSECT=YES / NO [,PREFIX=p]

DSECT=
Specifies whether a DSECT for the output area is to be generated or a data list.

> **YES**
> A DSECT is generated.
>
> **NO**
> A data list is created.

PREFIX=
Specifies a character string to be prefixed to the symbolic names of the DSECT/data list.

> **p**
> Prefix for the symbolic names. Length ≤ 2 characters.
> Default setting: p = TM.

Layout of the DSECT for the I/O area

```
           DTMODE DSECT=YES
1          #INTF REFTYPE=REQUEST,INTNAME=TMODE,INTCOMP=002
1 *---------------- P A R A M E T E R L I S T
1 TMODPL    DSECT
1           FHDR  UNIT=43,FUNCT=1,VERS=1
2          DS    0A
2          DS    0XL8                GENERAL OPERAND LIST HEADER
2          DC    AL2(43)             FUNCTION UNIT NUMBER
2          DC    AL1(1)              FUNCTION NUMBER
2          DC    AL1(1)              FUNCTION INTERFACE VERSION NUMBER
2          DC    X'FFFFFFFF'          Returncode is virgin
1 TMODTSN   DC    CL4' '                  TASK SEQUENCE NUMBER
1 TMODUSER  DC    CL8' '                  USER IDENTIFICATION NUMBER
1 TMODACCT  DC    CL8' '                  TASK ACCOUNT NUMBER
1 TMODTIME  DC    F'0'                    TASK CPU TIME
1 TMODPRIV  DC    AL1(0)                  TASK PRIVELEDGE CODE
1 TMODPRSA  EQU   1                       SYSTEM ADMINISTRATOR BIT
1 TMODPRUS  EQU   2                       USER BIT
1 TMODLLEN  DC    AL1(0)                  PHYSICAL LINE LENGTH (TERMINAL)
1 TMODVDT   DC    AL1(0)                  VIRTUAL DEVICE TYPE
1 TMODLINC  EQU   1                       LINE MODE CAPABILITY
1 TMODFORC  EQU   2                       FORMAT MODE CAPABILITY
1 TMODCMPC  EQU   4                       COMPATIBLE MODE CAPABILITY
1 TMODFYSC  EQU   8                       PHYSICAL MODE CAPABILITY
1 TMODEXLC  EQU   16                      EXTENDED LINE MODE CAPABILITY
1 TMODEOM   EQU   64                      EVANESCENT OUTPUT MESSAGES(VDU)
1 TMODTYPE  DC    AL1(0)                  TASK OR TERMINAL TYPE
1 TMODPRI   DC    AL1(0)                  TASK PRIORITY
1 TMODMSG   DC    X'00'                   MSG OPTIONS              :*
1 TMODMSGF  EQU   X'01'                     |
1 TMODMSGC  EQU   X'02'                     |
1 TMODMSGH  EQU   X'04'                     > SEE /OPTION COMMAND
1 TMODMSGT  EQU   X'08'                     |
1 TMODMSGL  EQU   X'20'                     |
1 TMODBUFS  DC    H'0'                    BUFFERSIZE
1 TMODPNAM  DC    CL8' '                  PROGRAM NAME
1 TMODNAME  DC    CL8' '                  JOB NAME FROM /LOGON
1 TMODPLL   EQU   *-TMODPL                 LENGTH OF PARAMETERLIST
```

Explanation of field contents

TMODTSN: task sequence or job number (TSN); 4 characters
TMODUSER: user ID; 8 characters
TMODACCT: account number; 8 characters
TMODTIME: CPU time consumed by the task, multiple of 100 microseconds; 8-digit decimal number. Values > 204800 sec are not output; this maximum value is repeated for subsequent calls
TMODPRIV: privilege status assigned to the task, where
X'01' ≙ task runs under the system administration ID (TSOS)
X'02' ≙ task runs under the (nonprivileged) caller's user ID
TMODLLEN: (physical) line length at the data display terminal; only when line mode is used
TMODVDT: characteristics of the data display terminal
TMODTYPE: type of data display terminal; X'00' if the job is a batch job, otherwise:
X'02' ≙ 8103 Printer Terminal
X'04' ≙ 8150 Data Display Terminal
X'11' ≙ TRANSDATA 8415, 8418
X'15' ≙ 8151 Data Display Terminal
X'16' ≙ 8152 Data Display Terminal
X'17' ≙ 8110 Printer Terminal
X'18' ≙ 8161 Data Display Terminal with 54 characters per line
X'19' ≙ 8161 Data Display Terminal with 64 characters per line
X'1A' ≙ 8161 Data Display Terminal with 80 characters per line
X'2C' ≙ 8162 Data Display Terminal
X'2D' ≙ 8160 Data Display Terminal
X'35' ≙ 9750 Data Display Terminal
X'4F' ≙ 9763 Data Display Terminal
TMODPRI: run priority of the task; 2-digit hexadecimal number
TMODBUFS: length of the physical I/O buffer of the terminal; 4-digit hexadecimal number. Not with batch jobs.
TMODPNAM: program name, if module was loaded using the static loader (ELDE); 8 characters. X'00...0', if module was loaded using the dynamic binder loader (DBL).
TMODNAME: job name from the SET-LOGON-PARAMETERS command.

ENACO – Enable contingency definition

General

Application area: Contingency processing; see page 110
Macro type: Type S, MF format **1**: standard/L/E form; see page 29

Macro description

The **ENACO** macro allows a routine to be defined as a contingency process with a special name. The contingency process must be defined before it can be specified in a **SOLSIG** or **POSSIG** macro. The **ENACO** macro provides an ID to be used in further macros which refer to the contingency process. A program can use up to 400 contingency processes simultaneously. The scope of the contingency process is local: use is restricted to the task of the calling program.

The addressing mode (AMODE) active at the time of the **ENACO** macro call must be identical with the one at the time the contingency routine is executed.

Macro format and description of operands

```
ENACO
--------------------------------------------------------------
 ⎧ CONAME=name                                 ⎫
 ⎩ CONAMAD=addr / (r) [,CONAMLN=length]        ⎭
 ,COADAD=addr / (r) ,COIDRET=addr / (r)
 [,COMAD=addr / (r) ] [,LEVEL=prio / (r)]
 [,PARMOD=24 /  31] [,MF=L / (E, ..)]
```

CONAME=
Specifies the name of the contingency process.

name
Name of the contingency process. 1 ≤ name length ≤ 54.

```
Name format:
1st character: letter, #, @;
2nd through to 54th character: any combination of characters from the
                               character set (A,...,Z,0,...,9,$,#,@).
The first blank (X'40') terminates the name.
```

CONAMAD=
Specifies the address of a field containing the name of the contingency process (for name format see CONAME).

addr
Symbolic address (name) of the field

(r)
Register containing the “addr” address value.

CONAMLN=
Specifies the length of the contingency process name if CONAMAD=... was specified.

length
Length of the name in bytes
Default setting:
– length attribute of the field specified with CONAMAD=...
– 54 bytes if CONAMAD=(r) was specified.

COADAD=
Specifies a field containing the start address of the contingency process. The field should be aligned on a word boundary.

addr
Symbolic address (name) of a field containing the start address

(r)
Register containing the “addr” address value.

COIDRET=
Specifies the address of a field where an ID of the contingency process is entered. The ID should be used in subsequent macros referring to the contingency process.
The field should be aligned on a word boundary.

addr
Symbolic address (name) of the field to hold the ID.

(r)
Register containing the “addr” address value.

COMAD=
Specifies a field containing a contingency message. The message is transferred to register R1 of the contingency process. A message specified here may be replaced by a message issued by a **SOLSIG** or **POSSIG** macro.

addr
Symbolic address (name) of the field containing the contingency message.

(r)
Register containing the “addr” address value.

LEVEL=
Specifies the priority level of the contingency process.

prio
Priority level of the contingency process. 1 ≤ prio ≤ 127.
Default value: prio = 1

(r)
Register containing the value of “prio”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the ENACO macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'04'	X'00'	The contingency process was defined for the calling task.
X'0C'	X'04'	The contingency process was already defined for the calling task. No action.
X'10'	X'04'	Invalid operands were specified. No action.
X'18'	X'04'	The maximum number of contingency processes which can be used simultaneously has been exceeded. No action.

For **examples**, see the section “Contingency processes” (page 110) and the description of the **POSSIG** macro (page 736).

ENAEI – Enable event item

General

Application area: Eventing; see page 94
Macro type: Type S, MF format **1**: standard/L/E form; see page 29

Macro description

The **ENAEI** macro allows a task to establish an event item. If an event item with the specified name already exists in the defined scope (established by an **ENAEI** macro call in another task), the macro only causes the event item to be assigned to the task of the calling program. Otherwise the event item is established and assigned by the system.
A call to this macro also makes available an ID for the event item; this ID can be used in subsequent calls, to speed up processing.
A program can use up to 2000 event items simultaneously.

Macro format and description of operands

ENAEI
{ EINAME=name { EINAMAD=addr / (r) [,EINAMLN=length] } ,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL ,EIIDRET=addr / (r) ,SOSIGQ=FIFO / LIFO [,PARMOD=24 / 31] [,MF=L / (E,..)]

EINAME=
Specifies the name of the event item.

name
Name of the event item. 1 ≤ name length ≤ 54.

```
Name format:
1st character: letter, #, @;
2nd through 54th character: any combination of characters from the
                            character set (A,...,Z,0,...,9,$,#,@).
The first blank (X'40') terminates the name.
```

EINAMAD=
Specifies the address of a field containing the name of the event item (for name format see above).

addr
Symbolic address (name) of the field.

(r)
Register containing the “addr” address value.

EINAMLN=
Specifies the length of the event item name if EINAMAD=... was specified.

length
Length of the name in bytes.
Default setting:

– length attribute of the field specified with EINAMAD=...
– 54 bytes if EINAMAD=(r) was specified.

SCOPE=
Specifies the scope of the event item (i.e. the participants authorized to use it).

LOCAL
the use of the event item is limited to the calling task.

GROUP
All the tasks with the same user ID as the calling task are participants.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.

The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

EIIDRET=
Specifies the address of a field where an ID of the event item is entered.

addr
Symbolic address (name) of the field to hold the ID. Field length = 4 bytes. The field should be aligned on a word boundary.

(r)
Register containing the “addr” address value.

SOLSIGQ=
Specifies the queueing method to be used for SOLSIG requests. All **ENAEI** macros having a given event item must define the same queueing method for SOLSIG requests.

FIFO
FIFO = first in - first out

LIFO
LIFO = last in - first out

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

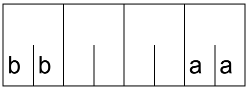
24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:



A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the ENAEI macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'04'	X'00'	Function executed: the event item was established by the system and assigned to the task of the calling program.
X'08'	X'00'	Function executed: an event item already established by the system was assigned to the task of the calling program.
X'0C'	X'04'	No action taken: the event item had already been assigned to the task of the calling program. The ID of the event item is supplied.
X'10'	X'04'	No action taken: invalid operands were specified.
X'18'	X'04'	No action taken: the maximum number of event items which can be used simultaneously has been exceeded.
X'1C'	X'04'	No action taken: event item specifications (FIFO and LIFO) do not agree for SOLSIG queueing (SOLSIGQ operand).

For **examples**, see the sections “Eventing” (page 106) and “Contingency processes” (page 118) and also the **POSSIG** (page 742) and **SOLSIG** macro description (page 816).

ENAMP – Enable memory pool

General

Application area:	Memory pools; see page 55
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

A memory pool (MP) is a memory area (class 6 memory) that may be used by several users together. The user who creates the memory pool defines its size (position), name and memory attributes.
Memory pools can be created in units of 64K and 1 Mb. Memory pools with units of 64K are always created below the 16-Mb boundary; on a long-term basis, such memory pools will no longer be supported on these servers. For this reason and reasons of degraded performance, users are advised against creating memory pools of 64K units. Information about the size of the memory pool and the number of memory pages allocated can be requested by means of the **MINF** macro.

Macro description

With the **ENAMP** macro, a user can create a memory pool or declare participation in an existing memory pool. The user is given an ID for the memory pool by the system. This ID can be used to increase the speed of processing in subsequent pool macros (**CSTMP, REQMP, RELMP, DISMP**); it can vary for different users.
When creating a new memory pool, the caller sets the following fixed pool attributes with **ENAMP**:

- name of the memory pool
- scope (authorized users: the caller only, all tasks under the user ID of the caller, all tasks from the user group of the caller, all tasks in the system)
- size
- start address (uniform or freely selectable for each user)
- location (below the 16 Mb boundary or freely selectable for each user)
- page management (resident or pageable)

The specified memory space is reserved in the address space of the caller with **ENAMP**. Pages are requested with **REQMP**.

The following points apply when using an existing memory pool:

- a memory pool can only be identified uniquely with its own name and the scope (SCOPE operand); (in subsequent calls the ID is sufficient).
- the caller must not specify any different pool attributes (see above) (it is best to use the default setting). The caller's access key must match the access key valid when the memory pool was established.
- any caller can request memory space with **REQMP** (as far as the size of the memory pool permits) and release memory space with **RELMP**.
- any authorized caller (CSTMP=YES in the user catalog) can define a lock to protect memory pages against access or release the lock (**CSTMP**).
- any caller can terminate participation in the memory pool by means of **DISMP**.

Notes

- the **WRCPT** macro and the NCHOLD and RESTART-PROGRAM commands are rejected if a task is using a memory pool.
- operand lists and input/output areas that are supplied dynamically should not be stored in memory pools (otherwise complicated synchronization measures are required).

Macro format and description of operands

```
ENAMP
-----------------------------------------------------------------------------------------

  { MPNAME=name                                       }
  { MPNAMAD=addr / (r) [,MPNAMLN=length / (r)]        },SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL

[,MPIDRET=addr / (r)]

,MODE=ANY / NEW / OLD

[, { BSIZE=size / (r) } ]
   { PSIZE=size / (r) }

[,LOC=BELOW]

[,PAGE=addr / (r)]

[,FIXED=YES]

,RES=NO / YES

,INHERIT=YES / NO

[,PARMOD=24 / 31]

[,MF=L / (E,..)]
```

MPNAME=
Defines the name of the memory pool. Note the connection with the SCOPE operand.

name
Memory pool name. 1 ≤ name length ≤ 54 characters.

```
Name format:
1st character: letter or special character #,@.
2nd through 54th character: any combination from the character set
                            (A,...,Z,0,...,9,$,#,@).
The first blank (X'40') terminates the name.
```

MPNAMAD=
Defines the address of the field with “name”. Note the connection with the SCOPE operand.

addr
Symbolic address (name) of the field.

(r)
Register containing the address value of the field.

MPNAMLN=
Defines the length of the name specified under MPNAMAD. If it is not specified: length attribute of the “addr” field or, if MPNAMAD=(r) was specified, 54 bytes.

length
Length in bytes.

(r)
Register containing the length.

SCOPE=
Specifies the scope (authorized participants) for the memory pool.
The operand allows a unique identifier to be assigned to the memory pool, and must be used in conjunction with the operands MPNAME and MPNAMAD. Note the default value!

LOCAL
the use of the memory pool is limited to the calling task.

GROUP
All the tasks with the same user ID as the calling task are participants.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.

The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

MPIDRET=
Defines the ID for the memory pool. The ID is returned to the user by the system after execution of the macro and can be used in subsequent macros (**REQMP, RELMP, DISMP, CSTMP**) to identify the pool uniquely (the use of the ID increases the speed of processing). All memory pool users are given their own ID by **ENAMP**. An exception are “inheritable” memory pools, where the “son task” can use the same ID as the “parent task”.

addr
Symbolic address (name) of the field containing the ID.
Field length = 4 bytes.

(r)
Register containing the address value of the field.

MODE=
Specifies whether the caller wants to create a new memory pool or use one that already exists. If the latter is the case, it should be noted that a memory pool can only be identified uniquely via its name and scope (SCOPE operand).

ANY
the caller wants to use the specified memory pool if it exists; if not, a memory pool with the specified attributes is created for the caller.

NEW
The caller wants to generate a new memory pool with the specified attributes.
The call is rejected if a memory pool with this name and SCOPE already exists.

OLD
The caller wants to use an existing memory pool. The call is rejected if

- the memory pool does not exist,
- the specified pool attributes do not correspond to those determined at pool creation
- the memory pool was generated with SCOPE=LOCAL.

BSIZE=
Defines the size of the memory pool (in 4K units = memory pages). The memory pool is created as a contiguous memory area as specified by means of the operands PAGE and/or LOC and, depending on the addressing mode, below or above the 16-Mb boundary. BSIZE cannot be specified if the PSIZE operand was specified at memory pool creation.

size
Number of memory pages (4K units). Specification of size = 0 is illegal and results in the macro call being rejected. The size of the memory pool may be rounded by the operating system for performance reasons (see Notes).

(r)
Register containing the number of memory pages
Default setting:

- For reasons of compatibility, a (new) memory pool is created using the default value for the PSIZE operand if neither BSIZE nor PSIZE is specified explicitly.
- Current value for existing memory pool.

Notes

- The size of the memory pool may be rounded by the operating system for performance reasons as follows:
  The memory pool is created with units of 1 Mb and aligned on a 1-Mb boundary. It is rounded to a multiple (n) of 1 Mb units (size of MP = n * 1Mb ≥ size * 4K).
- Rounding is guaranteed to be part of the functional scope of BS2000 Version 9.0 only.

PSIZE=
Defines the size of the memory pool in 64K units. The memory pool is created as a contiguous memory area below the 16-Mb boundary and aligned on a 64K boundary. PSIZE cannot be specified if the BSIZE operand was specified at memory pool creation.

size
Number of memory segments with 64K each. size=0 is illegal; the macro is rejected in this case.

(r)
Register containing the value for “size”.
Default value (only if operand BSIZE was not specified at memory pool creation):

- size = 1 for a memory pool to be created,
- current value for an existing memory pool.

Note
On a long-term basis, the PSIZE operand and memory pools with units of 64K will no longer be supported.

LOC=
Specifies that part of the address space where the memory pool is to be located. Specification of this operand is useful only in conjunction with 31-bit addressing and if the BSIZE operand is specified.

BELOW
The memory pool is created in the caller's address space below the 16-Mb boundary.

Note
A memory pool created with LOC=BELOW but without a fixed start address may, in another participant's own address space, also be placed above the 16-Mb boundary.

PAGE=
Specifies the start address of the memory pool in the caller's address space. The start address should be aligned as follows:

- on a 64K boundary, if the memory pool consists of 64K segments (PSIZE operand).
- on a 1-Mb boundary, if the memory pool consists of 1-Mb segments (BSIZE operand).

Default setting: The first sufficiently large and contiguous area that begins on a 64K or 1-Mb boundary is selected.
In 31-bit addressing mode, the memory pool is located above the 16 Mb boundary unless LOC=BELOW is specified. If the memory pool was created with FIXED=YES, the fixed start address is binding for all future users.
Default value in this case: the start address is assumed (the memory pool is placed in the address space of the caller in the same way as in the address space of the one that created the memory pool). The start address is submitted to the caller in register R1.

addr
Start address.

(r)
Register containing the address value “addr”.

Note
Wherever possible, the PAGE operand should not be used. Should it be indispensable, the user should previously inquire about the size and location of the class 6 memory by means of the **MINF** macro.

FIXED=YES
The memory pool has the same virtual start address for all users (from this address onwards, it is placed in the caller's address space).

Notes

- if FIXED=YES is not specified (by the first caller), future users may each specify a different start address in their address space.
- a caller that wants to use an existing memory pool cannot revoke the FIXED=YES specification.
- specification of this operand is useful only for the caller that created the memory pool. If it is not specified at memory pool creation, any subsequent specification of FIXED=YES by a pool participant will be rejected.

RES=
Specifies whether the memory pages of the pool are to be pageable or resident. This attribute is determined by the caller that creates the pool (the maximum number of resident memory pages is set in user catalog).

NO
The memory pages are to be pageable.

YES
The memory pages are to be resident.

Note

In the case of RES=YES, the specification for the PSIZE/ BSIZE operand is not checked against the specification for the RESIDENT-PAGES operand in the START-PROGRAM or LOAD-PROGRAM command. The check is not carried out until memory space is occupied with **REQMP**.

INHERIT=
Specifies the inheritability of a memory pool when the participating user task (“parent task”) generates a new son task: (“fork()”). This operand is not supported if a local and resident class 6 memory pool was specified or PARMOD=24 was set. See also note on page 465.

<u>YES</u>
The memory pool should be inheritable i.e. in the event of a fork(), the son task is implicitly linked to a non-local memory pool of the parent task.
For a local (non-resident) memory pool, the memory space of the parent task is copied to the son task. This means that the memory space for each task is locally available and the task is free to decide on the contents of the memory pool and the functions to be executed.

NO
In the event of a fork(), the son task is not linked to the memory pool of the parent task. The memory area is not assigned to the son task.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists use 24-bit addresses.
(Address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists use 31-bit addresses
(address space ≤ 2 Gb) and start with the standard header.

Notes on memory pool inheritability

- For inheritable, *local* memory pools, a new memory pool is set up implicitly for the “son task” (after task generation by the parent task: “fork()”). This has the same properties (name, access rights etc.) and contents as the memory pool of the parent task at the point when the son task was generated (implemented via the copy-on-write mechanism). This enables both parent and son tasks to have shared access to the contents of the memory pool, while any changes effected by them apply only locally within their “own” memory pool.
- For inheritable, *non-local* memory pools, inheritance is effected via an implicit **ENAMP** call to the existing memory pool during the fork(). This means that parent and son tasks operate in the same memory area.
- The ID of the memory pool (MPIDRET operand) is inherited from the the parent task by the son task.
- However, the allocation for requesting resident memory pages and - for a local memory pool - the resident memory pages priorized with **CSTAT** PAGE=NO, are not inheritable.

Return information and error flags

The virtual start address of the memory pool is stored in register R1 after the function has been carried out.

R15:	b	b	0	0	0	0	a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the ENAMP macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'04'	X'00'	Normal execution; a new memory pool has been established (MODE=NEW/ANY).
X'08'	X'00'	Normal execution; the caller is the new user of the specified memory pool (MODE=OLD/ANY).
X'04'	X'04'	Function was not executed; memory pool does not exist (MODE=OLD).
X'08'	X'04'	Function was not executed. The macro refers to an existing memory pool: – the caller is already a memory pool user (MODE=OLD/ANY). Register R1 contains the start address of the memory pool and MPIDRET its ID . – the memory pool already exists (MODE=NEW). – different pool attributes were specified (MODE=OLD/ANY): – PSIZE/BSIZE – PAGE (in the case of a fixed memory pool: the start address of the MP is returned in register R1) – LOC (in the case of a fixed memory pool) – FIXED – RES
X'14'	X'04'	Function was not executed; insufficient free memory space – in the caller' s address space – in the address space below the 16-Mb boundary (in conjunction with LOC=BELOW or PSIZE=...)
X'18'	X'04'	Function was not executed; invalid memory address: – the start address or an address in the specified area is outside the address space of the caller. – the start address or an address in the specified area is on or above the 16-Mb boundary and LOC=BELOW or PSIZE=... was specified. – the start address is not aligned on a 64K/1-Mb boundary – the specified address space is not completely free.

X'bb'	X'aa'	Meaning
X'1C'	X'04'	Function was not executed. Operand error: – invalid address of the operand list – error in structure of operand list – invalid addresses for MPNAMAD or MPIDRET in the operand list – designation of the memory pool: – the name contains invalid characters – invalid length specification (MPNAMLN) – the name (MPNAME, MPNAMAD) is not specified – MPNAME and MPNAMAD are specified – MPNAMLN specified, but MPNAMAD omitted – SCOPE specified, but MPNAME/MPNAMAD omitted – invalid specifications for SCOPE/MODE/BSIZE/PSIZE/LOC/FIXED/RES – the MP name and the field in which the ID is transferred overlap – invalid register (R1) specified – PARMOD=24 specified in conjunction with 31-bit addressing mode (AMODE31). – SCOPE=USER_GROUP was specified, although SRPM is not available in the system. – With an ENAMP call for a memory pool which already exists, the current access key does not match the access key valid when the pool was created. – INHERIT=YES was specifed, although the memory pool in question is local and resident or PARMOD=24 was specified.
X'20'	X'04'	Function was not executed. Owing to memory saturation, the macro cannot be executed at the moment. A subsequent macro may be successful.

31-bit interface:

- In the event of errors in the alignment or initialization of the standard header, the return codes X'0001FFFF' / X'0003FFFF'/ X'0004FFFF' are additionally transferred in register R15; see the table “Standard return codes” on page 43.
- No return codes are transferred in the standard header.

ENASI – Enable serialization item

General

Application area:	(Task) serialization; see page 91
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

ENASI generates either the 24-bit or the 31-bit interface, depending on the specification. In the event of macro chaining, all macros chained must make use of the same interface (either 24-bit or 31-bit interface).

Macro description

The **ENASI** macro creates a serialization item for the calling program's task. If a serialization item with the specified name has already been established in the defined scope (by an **ENASI** macro of another task), the macro call only causes the serialization item to be assigned to the task of the calling program. If there is no serialization item, one is established by the system and assigned to the task of the calling program. This macro also returns an ID for the serialization item; this ID can be used in later calls to speed up processing.

A program can use up to 2000 serialization items simultaneously. The CONTINU operand allows up to 255 **ENASI** macros to be chained.

Macro format and description of operands

ENASI
{ SINAME=name { SINAMAD=addr / (r) [,SINAMLN=length] } ,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL ,SIIDRET=addr / (r) ,CONTINU=NO / YES [,PARMOD=24 / 31] [,MF=L / (E,..)]

SINAME=name
Specifies the name of the serialization item. The name consists of a character string comprising 1 to 54 bytes. The first blank (X'40') terminates the name. The characters may be letters, digits and the special characters $, # and @. The first character must not be a digit or the $ character.

SINAMAD=
Specifies the name of the serialization item.
The rules governing the name format are specified in the SINAME operand description.

addr
Symbolic address of the field containing the name.

(r)
Register containing the address.

SINAMLN=
Specifies the length in bytes of the serialization item name. The length must be at least 1 byte and not more than 54 bytes.
If the operand is missing, the length attribute of the SINAMAD operand is assumed if SINAMAD=addr is specified; if SINAMAD=(r), the maximum length (54) is assumed.

length
Length of the serialization item name.

SCOPE=
Specifies the scope of the serialization item (i.e. the participants authorized to use it).

LOCAL
The use of the serialization item is limited to the calling task.

GROUP
All the tasks with the same user ID as the calling task are participants.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.

The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

SIIDRET=
Specifies the ID of the serialization item.
The ID can be used in other macros (**ENQAR**, **DEQAR**, **CHKSI** and **DISSI**) referring to the specified serialization item in order to speed up processing.

addr
Symbolic address (name) of a 4-byte field used for returning the ID to the calling program.

(r)
Register containing the address value “addr”.

CONTINU=
This operand allows up to 255 **ENASI** macros to be chained.

NO
This is the last (or only) macro of a sequence.

YES
YES indicates that another **ENASI** macro follows this macro.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Notes on the macro call

- The ID of the serialization item passed to the calling program can be used in other macros instead of the name in order to speed up processing.
- An explicit enable for a serialization item is required only if the user desires to speed up processing by using the ID. Otherwise the user can employ the implicit enable function (see the **ENQAR** and **DEQAR** macros).
- A task can use a serialization item only if an associated explicit or implicit enable function for this task was processed.
- A second **ENASI** macro call (explicit or implicit) issued in a program for a serialization item that was specified earlier in an explicit enable and then disabled again (see the **DISSI** macro), does not necessarily assign the same ID to this item as to the first **ENASI** macro call.
- If the same serialization item name is specified in two **ENASI** macros with different scopes, the system will process two different serialization items.
- The scope of an ID is the same as the scope of the associated name.

- When the list form of the macro (MF=E) is used, the following should be noted: Only one macro with MF=E is required for execution, regardless of whether the macro applies to one request or to a series of requests. In the case of a series of requests, the operand list is generated through macro chaining (MF=L) by means of the CONTINU operand.

Return information and error flags

Register 1 contains the operand list address.

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the ENASI macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'04'	X'00'	All enable macros were executed: At least one new serialization item was established.
X'08'	X'00'	All enable macros were executed: The use of at least one serialization item was enabled.
X'0C'	X'04'	Not all enable macros were executed: At least one serialization item has already been used by the task of the calling program.
X'10'	X'04'	Not all enable macros were executed: Invalid operands were specified: – invalid address – invalid length – invalid name – scope or CONTINU value undefined.
X'18'	X'04'	Not all enable macros were executed: The maximum number of concurrently used serialization items was exceeded.

ENQAR – Enqueue access request

General

Application area: (Task) serialization; see page 91
Macro type: Type S, MF format **1**: standard/L/E form; see page 29

ENQAR generates either a 24-bit or a 31-bit interface, depending on the specification. In the event of macro chaining, all macros chained must make use of the same interface (either 24-bit or 31-bit interface).

Macro description

This macro requests access to the specified serialization item. The access request is entered in the queue of the serialization item and the program is placed in the wait state until it is the first in the queue. The program is then continued and uses the serialization item until a **DEQAR** macro is issued for this serialization item.
If there is no serialization item with the specified name in the defined scope, the serialization item is established and an implicit enable function (see the **ENASI** macro) is performed.
The COND operand enables the user to specify whether the access request is to be performed unconditionally, or only if it can be satisfied immediately. The time for which a program will wait for an access request to be honored can be limited by means of the LIFETIM operand.
The CONTINU operand allows up to 255 **ENQAR** macros to be chained. Such a chain of macros will be processed only if all individual requests can be honored at the same time.

Macro format and description of operands

```
ENQAR
--------------------------------------------------------------------------------
{ { SINAME=name                                  }                                                  }
{ { SINAMAD=addr / (r) [,SINAMLN=length]         },SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL        }
{                                                                                                   }
{ SIID=adr / (r)                                                                                    }

,CONTINU=NO / YES

,COND=UNCOND / IMMED

[,LIFETIM=sec / (r)]

[,PARMOD=24 / 31]

[,MF=L / (E,..)]
```

SINAME=name
Specifies the name of the serialization item. This specification is unique only in conjunction with the SCOPE operand.

SINAMAD=
Specifies the address of the serialization item name. This name is unique only if the SCOPE operand is also specified.

addr
Symbolic address of the field containing the name.

(r)
Register containing the address.

SINAMLN=
Specifies the length in bytes of the serialization item name. The length must be at least 1 byte and not more than 54 bytes.
If the operand is missing, the length attribute of the SINAMAD operand is assumed if SINAMAD=addr is specified; if SINAMAD=(r), the maximum length (54) is assumed.

length
Length of the serialization item name.

SCOPE=
Specifies the scope of the serialization item (i.e. participants authorized to use it).

LOCAL
The use of the serialization item is limited to the calling task.

GROUP
All the tasks with the same user ID as the calling task are participants.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.
The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system.

This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

SIID=
Specifies the ID of the serialization item. This ID is supplied to the user by the **ENASI** macro. If the ID is used instead of the name of the serialization item, processing is speeded up. The ID identifies the item uniquely.

addr
Symbolic address of a 4-byte field containing the ID.

(r)
Register containing the address.

CONTINU=
This operand allows up to 255 **ENQAR** macros to be chained.

NO
This is the last (or only) macro of a sequence.

YES
YES indicates that another **ENQAR** macro follows behind this macro.

COND=
Defines access request processing. If the request can be satisfied immediately, it is satisfied; if it cannot be satisfied immediately, the request is entered in the queue of the referenced serialization item. In this case, the calling program must wait until the complete request can be processed or until the waiting time specified by means of the LIFETIM operand has elapsed.
This operand may be used only in the last macro of a series of **ENQAR** macros chained by means of CONTINU; nevertheless it applies to the whole series.

UNCOND
The request is not subject to any conditions.

IMMED
The access request is processed only if the complete request can be satisfied immediately.

LIFETIM=
Specifies the time in seconds the task is to wait for the access requests to be processed. A return code indicates whether the request was satisfied or whether the maximum waiting time has elapsed.
The operand may be used only in the last macro of a series of **ENQAR** macros chained by means of CONTINU; nevertheless it applies to the whole series.

sec
Time in seconds. 1 sec ≤ wait time ≤ 43200 sec.
Processing precision for this macro is +10 seconds. Default setting: 600 seconds.

(r)
Register containing the time in seconds.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

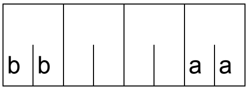
Notes on the macro call

- An implicit enable is performed when the calling task has not yet requested an explicit or implicit enable (see the **ENASI** macro) for this serialization item. However, an implicit enable does not cause an ID to be supplied.
- Each further explicit or implicit enable macro issued in a program for a serialization item that was specified earlier in an explicit enable and then disabled (see the **DISSI** macro) does not necessarily assign the same ID to this serialization item as the first enable macro.
- An enable performed as part of an **ENQAR** macro and the entry in the queue form one operation.
- If the same serialization item name was specified in two enable macros with different scopes, the system will process two different serialization items.
- When the list form of the macro (MF=E) is used, the following should be noted:
  Only one macro with MF=E is required for execution, regardless of whether the macro applies to one request or to a series of requests. In the case of a series of requests, the operand list is generated through macro chaining (MF=L) by means of the CONTINU operand.

Return information and error flags

Register R1 contains the operand list address.

R15:



A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the ENQAR macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'04'	X'00'	All enqueue requests were satisfied: At least one serialization item was established and assigned to the task of the calling program.
X'08'	X'00'	All enqueue requests were satisfied: At least one serialization item was assigned to the task of the calling program.
X'0C'	X'00'	All enqueue requests were satisfied: All serialization items were already assigned to the task of the calling program.
X'3C'	X'00'	All enqueue requests were satisfied: A DEQAR macro, issued by another task that was not the “holder” of the serialization item, has already been processed. This DEQAR macro was processed because HOLDER=ANY was specified. This return code has priority over all others with RS=X' 00' .
X'10'	X'04'	Not all enqueue requests were satisfied: Invalid operands were specified: – invalid address (e.g. address within a DSECT) – invalid length, invalid name, invalid LIFETIM value – SCOPE, CONTINU or COND value undefined – COND or LIFETIM specification entered for an item that is not the last in a series chained with CONTINU.
X'14'	X'04'	Not all enqueue requests were satisfied: An invalid ID was specified.
X'18'	X'04'	Not all enqueue requests were satisfied: The maximum number of serialization items which can be used simultaneously was exceeded.
X'1C'	X'04'	Not all enqueue requests were satisfied: The complete request could not be satisfied – immediately (if COND=IMMED) – within the waiting time (if COND=UNCOND).
X'24'	X'04'	Not all enqueue requests were satisfied: The calling program is already using the requested serialization item.
X'40'	X'04'	Not all enqueue requests were satisfied: No class 5 memory was available for processing the request.
X'44'	X'04'	Not all enqueue requests were satisfied: A disable request for at least one serialization item was issued (by a contingency process with higher priority), while the enqueue request was still pending.

ENTER – Initiate ENTER job

General

Application area:	Starting, interrupting and terminating; see page 72
Macro type:	Type S, MF format **1**: standard/L/E/C/D form; see page 29

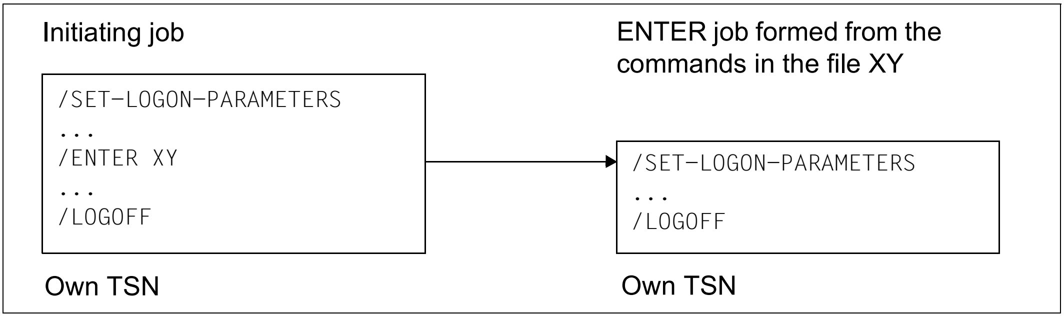
Macro description

The **ENTER** macro allows the **ENTER-JOB** command to be issued via the macro command language processor (MCLP) without interrupting program mode (see section “Macro Command Language Processor macros” on page 45). Messages concerning command processing are output to SYSOUT and may additionally be transferred to an area of the calling program. The ENTER-JOB command allows a batch job that is stored in an (ENTER) file to be transferred to the operating system for processing.
The (ENTER) file is a cataloged file or a library element. The ENTER-JOB command can be issued in both program mode and command mode (see also the “Commands” manual [19]). The new job receives its own TSN and is executed in its own task - independently of the calling task. The specifications in the ENTER-JOB command designate the (ENTER) file, identify the caller (access authorization and accounting) and characterize the job and logging for the job run.

The specifications on access authorization are checked against the entry in the user catalog; further specifications regarding the job class and job attributes (job priority, run priority, system resources) are also checked against the entry in the job class definition. These entries may be accessed by the user via the SHOW-USER-ATTRIBUTES or SHOW-JOB-CLASS commands.
If the entries for PRIORITY and NTL (No Time Limit) are not identical in the user catalog and the job class definition, the value that is better for the user is accepted.
The PRIORITY and MSG operands only continue to be supported for reasons of compatibility. Instead, the RUN-PRIO operand - or the RUN-PRIO, START= IMMEDIATELY (for PRIORITY=(p,EXPRESS) and LOG operands together - should be used.

The operand list specified in the command operands must be specified as a string enclosed in single quotes ('pathname [,userid1, ... ,START=..]'). Single quotes that are a part of a string that occurs within this operand list (e.g. specification of the operand HOST='hostid') must be specified twice to prevent them from being interpreted as special characters.

An (ENTER) file begins with the SET-LOGON-PARAMETERS command and ends with the EXIT-JOB command. The operands in the SET-LOGON-PARAMETERS command are not interpreted.



```
/SET-LOGON-PARAMETERS
...
/ENTER XY
...
/LOGOFF
```

```
/SET-LOGON-PARAMETERS
...
/LOGOFF
```

Figure 24: Initiating an ENTER job

Macro format and description of operands

```
ENTER

'<text>'
where <text> consists of the following operands (single quotation marks must be specified twice):
     pathname
     [,userid1,accountno[,password]]
     [,FROM-LOGON=YES / NO]
     [,FPASS=password]
     [,CRPASS=password]
     ,ERASE=NO / YES
     [,HOST=*ANY / ' 'hostid' ' / jvname1] [,CAT=' 'catid' ' / jvname2]
     [,JOB-CLASS=*STD / jobclass]
     [,MONJV=jvname] [,JVPASS=password]
     [,JOB-PRIO=STD / jprio]
     [,RERUN=NO / YES]
     [,FLUSH=NO / YES]

                ⎧ STD                                                  ⎫
                ⎪ SOON                                                 ⎪
                ⎪                                                      ⎪
                ⎪ WITHIN(⎧ HOURS=hours[,MINUTES=minutes]  ⎫)           ⎪
                ⎪        ⎩ [HOURS=hours, ]MINUTES=minutes ⎭            ⎪
     [,START=   ⎨ AT([DATE=yy-mm-dd, ]TIME=hh : mm)                    ⎬ ]
                ⎪ EARLIEST([DATE=yy-mm-dd, ]TIME=hh : mm)              ⎪
                ⎪ LATEST([DATE=yy-mm-dd, ]TIME=hh : mm)                ⎪
                ⎪ AT-STREAM-STARTUP                                    ⎪
                ⎩ IMMEDIATELY                                          ⎭
```

```
ENTER (cont.)

          [,REPEAT=  { STD                                                  } ]
                     { NO                                                   }
                     { PERIOD({ HOURS=hours[ ,MINUTES=minutes]  })          }
                     {        { [HOURS=hours, ]MINUTES=minutes  }           }
                     { DAILY                                                }
                     { WEEKLY                                               }
                     { AT-STREAM-STARTUP                                    }
          [,CALENDAR=' 'pathname' ' ,SYMDATE=symdatname]
          ,LIMIT=STD / number / (DATE=yy-mm-dd,TIME=hh:mm)
          [,RUN-PRIO=STD / rprio]
          [,TIME=STD / NTL / t]
          ,PROTECTION=NONE / CANCEL
          [,PRINT= STD / NO-LIMIT / number
          [,LOG=(LISTING=NO / YES)]
          [,JOB-PAR=*NO / ' 'attributes' ']
          [,PRIORITY=p / ([p],EXP[PRESS])]
          ,MSG=[F / C] [L] [H]
[,addr / (r)]
[,PARMOD=24 / 31]
[,MF=C / D / L / (E,..)]
```

The 'pathname' variable stands for: [:catid:] [$userid.]{ filename / library(element) }

where:

catid
Catalog ID of the pubset on which the file is stored. Default value: the catalog ID allocated to the user ID in the user catalog.

userid
User ID to which the file is allocated.
Default value: user ID which executes the macro call.

i When `pathname` is specified without a catalog ID and user ID and it is not cataloged under the user's own ID, the system attempts to access a file or library of the same name under the default system ID ("Secondary Read" function, see the "Introductory Guide to DMS" [8]).

filename
Name of the cataloged file with the batch job.

- `filename` can also be the name of a temporary file (see the “DMS Introductory Guide” [8])
- A file generation or a file generation group cannot be specified
- The specification of a file group (“file(group)” format) is only permissible for tape files
- The file must be shareable if it does not belong to the own user ID

library
Name of a PLAM library on disk (see the “LMS” manual [29]).

(member)
Name of the library element with the batch job.

The expression library(member) without a catalog ID and user ID may be up to 41 characters long. With a full path name including catalog ID and user ID it may be up to 54 characters long.

userid1
User ID for the ENTER job to be initiated or `*FROMCA` if the macro call is issued from a user ID with the OPERATING privilege.

***FROMCA**
The user ID of the caller is used.

accountno
Account number for the ENTER job.

The `userid1` and `accountno` operands may only be specified or omitted together in the ENTER macro. If they are omitted in the ENTER macro, the values in the SET-LOGON-PARAMETERS command of the executing job are assumed.

password
Password for the user ID `userid1`.
password is a string with a length of 8 bytes (c-string) or 16 bytes (x-string). Because `password` is a string within a string, the single quotes must be specified twice. The password is not logged on SYSOUT, i.e. it does not appear in the printout of the ENTER job.
If the `userid1`, `accountno` and `password` operands are omitted, they are, in the case of LOGON=YES, taken over from the SET-LOGON-PARAMETERS command of the ENTER file. Otherwise they are taken over from the SET-LOGON-PARAMETERS command of the initiating job.

FROM-LOGON=

Specifies whether or not the operands of the SET-LOGON-PARAMETERS command with which the ENTER file begins are to be evaluated (as with an ENTER-JOB command at the console).

YES

This operand can only be specified for a macro call from a user ID with the OPERATING privilege.

The operands of the SET-LOGON-PARAMETERS command in the ENTER file are evaluated. However, entries in the ENTER macro have priority, i.e. a value specified in the SET-LOGON-PARAMETERS command becomes effective only if the corresponding operand in the ENTER macro is **not** specified.

NO

The operands of the SET-LOGON-PARAMETERS command in the ENTER file are not evaluated.

FPASS=

Designates the execute or write password for the ENTER file: write password when ERASE=YES is specified, otherwise the execute password. The operand allows access to the ENTER file if the password specified matches the password that protects this file.

password

Password for accessing the ENTER file.

`password` is a string with a length of 4 bytes (c-string) or 8 bytes (x-string). Because `password` is a string within a string, the single quotes must be specified twice. The password is not logged on SYSOUT, i.e. it does not appear in the printout of the ENTER job.

CRPASS=

Designates the password with which the ENTER file is encrypted. The operand allows access to the encrypted ENTER file if the password specified matches the password that protects this file.

password

Password for accessing the ENTER file.

`password` is a string with a length of 8 bytes (c-string) or 16 bytes (x-string). Because `password` is a string within a string, the single quotes must be specified twice. The password is not logged on SYSOUT, i.e. it does not appear in the printout of the ENTER job.

ERASE=
Specifies whether the ENTER file is to be erased when the job is terminated.

NO
The ENTER file is not to be erased when the job is terminated.

YES
The ENTER file is to be erased at the end of the ENTER job.

In spite of ERASE=YES, the file is not deleted if

a) the file is a library member

b) the job issuer is not the file’s (co-) owner

c) the job is abnormally terminated

d) the job is terminated by an EXIT-JOB (MODE=*ABNORMAL), CANCEL-JOB or SHUTDOWN command

Cases c) and d) do not apply to a file on private disk, a temporary file or a file cataloged under the caller's user ID if the ENTER job is to run under a different user ID. In all these cases, the file is deleted once the S.IN. file has been created.

HOST=
Specifies the target system on which the job is to be run.
Only available to users of the software products “HIPLEX MSCF” [26] and “JV” [22].

***ANY**
The job can run on any target system.

"hostid"
ID of the target system.

jvname1
Job variable containing the host ID. The syntax for jvname1 must comply with the rules for a **GETJV** operation (see “JV” manual [22]).

CAT=
Specifies the target system via the specified catalog ID. The job is directed to the system to which the specified catalog is assigned.
Available only to users of the software products “HIPLEX MSCF” [26] and “JV” [22].

"catid"
Catalog ID.

jvname2
Job variable containing the catalog ID. The syntax for jvname2 must comply with the rules for a **GETJV** operation (see “JV” manual [22]).

JOB-CLASS=
Designates a job class to which the job is to be assigned.
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-LOGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value *STD.
The authorization for the various job classes can be queried with the SHOW-USER-ATTRIBUTES or SHOW-JOB-CLASS commands.

***STD**
The job class is the (standard) job class preset for the user or the system.

jobclass
Name of the job class.

MONJV=
Denotes a job variable that monitors the job.
If the operand is not specified, when FROM-LOGON=YES the value from the SET-LOGON-PARAMETERS command in the ENTER file applies, otherwise the ENTER job is started without MONJV.

Users can address their job via this job variable. During the job run, the operating system allocates the following value to the job variable:

$S	job in job queue
$R	job being processed
$T	job terminated normally
$A	job aborted
$M	job exported by means of /MOVE-JOBS

Only available to users of the “JV” software product [22].

jvname
Name of the job variable.

JVPASS=
Denotes a password that authorizes access to the monitoring job variable.
JVPASS is ignored if MONJV was not specified.

password
Password for the job variable jvname.
password = string with a length of 4 bytes (c-string) or 8 bytes (x-string). Because “password” is a string within a string, the single quotes must be specified twice. The password is not logged on SYSOUT, i.e. it does not appear in the printout of the ENTER job.

JOB-PRIO=
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-LOGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value STD.

STD
The standard value for the job class is assumed.

jprio
Job priority. MAXIMUM ≤ jprio ≤ 9. The lower the value, the higher the job priority (urgency). The value for MAXIMUM is set in the job class definition and can be accessed with the SHOW-JOB-CLASS command.

RERUN=
Specifies whether the job is to be rerun in the next BS2000 session if its execution was interrupted by serious system errors or by the end of the system run.
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-LOGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value NO.

NO
No rerun of the job.

YES
The job is rerun.

FLUSH=
Specifies whether the job is removed from the job queue if it has not been processed before the end of the system run (SHUTDOWN).
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-LOGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value NO.

NO
The job remains in the queue. The next system run must be initialized with a warm or selective start.

YES
The job is removed from the queue. Job control with RERUN/FLUSH:

Job control using RERUN/FLASH

- if FLUSH=YES and RERUN=YES were specified and the job was interrupted during the previous system run, FLUSH=NO is assumed in the next system run. This ensures that the job remains in the job queue even if it is not started in this system run.
- a monitoring job variable is set to $S if the job is repeated.
- RERUN and FLUSH are not evaluated if the job is repeated.

START=

Denotes a time (period of time) for the start of the job.
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-LOGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value STD.

STD

The standard value for the selected job class is assumed.

SOON

The job is to be started as soon as possible (in line with its priority).

IMMEDIATELY

The job is to be started immediately.

WITHIN(...)

The job is to be started within the specified time (hours/minutes).
0 ≤ hours ≤ 23; 0 ≤ minutes ≤ 59.

AT(...)

The job is to be started at precisely the specified date and time.

```
DATE=yy-mm-dd  Date (yy = year, mm = month, dd = day)
TIME=hh:mm     Time of day (hh = hours, mm = minutes)
```

- Hyphens and colons in DATE= and TIME= must be specified;
  for example: 31 May 2012 at 15.08 is AT (DATE=12-05-31, TIME=15:08).
- The following applies to TIME: 00 ≤ hh ≤ 23; 00 ≤ mm ≤ 59.
- If the two-digit year specification is less than 80, it is interpreted as 20yy, entries equal to or greater than 80 are interpreted as 19yy.

EARLIEST(...)

The job is to be started no earlier than at the specified time/date.

```
DATE=yy-mm-dd  Date (yy = year, mm = month, dd = day).
TIME=hh:mm     Time of day (hh = hours, mm = minutes).
```

See the START=AT(...) operand.

LATEST(...)

The job is to be started no later than at the specified time (date/time).

```
DATE=yy-mm-dd  Date (yy = year, mm = month, dd = day).
TIME=hh:mm     Time of day (hh = hours, mm = minutes).
```

See the START=AT(...) operand.

AT-STREAM-STARTUP

The job is to be started after the startup of the job scheduler.

Specification of the start values SOON, IMMEDIATELY, WITHIN, AT, EARLIEST, LATEST and AT-STREAM-STARTUP is only valid if they are authorized in the job class definition; see the SHOW-JOB-CLASS command.

REPEAT=
Denotes a period of time after which the job is to be started at regular intervals. The repetition is regarded as a job sequence. J(0) denotes the first job run, J(1) denotes the first repetition, ..., and J(n) denotes the nth repetition. The repetition J(i+1) is created with the start of the job J(i), where (i ≥ 0).
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-LOGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value STD.

STD
The standard value for the selected job class is assumed.

NO
The job is not repeated.

DAILY
Daily repetition at the time specified with START.

WEEKLY
Weekly repetition at the time specified with START.

PERIOD(...)
Repetition after the specified time interval (in hours and minutes).
0 ≤ hours ≤ 23; 0 ≤ minutes ≤ 59.

AT-STREAM-STARTUP
Repetition after every startup of the job scheduler.

Notes

- Specification of the repeat values NO, DAILY, WEEKLY, PERIOD and AT-STREAM-STARTUP is only valid if they are also authorized in the job class definition; (see the SHOW-JOB-CLASS command).
- The RERUN and FLUSH operands are not evaluated if a job is repeated. The job is restarted in the next repetition run.
- The ith repetition of a job (i ≥ 1) is not started until the (i-1)th execution has been terminated.
- The abnormal termination of the current job J(i) has no effect on the start of J(i+1); (i ≥ 0).
- Abnormal termination of the complete job: both the job that is currently running J(i) and the subsequent job J(i+1) must be terminated abnormally, (i ≥ 0); (CANCEL-JOB command or make the job the last job of the repeat sequence with the command MODIFY-JOB tsn, REPEAT=NO).

CALENDAR=
The start time of the job and any repetitions are specified by a symbolic date, which is defined in a calendar file (calendar job).
The CALENDAR and SYMDATE operands must be specified together.
If CALENDAR and SYMDATE are specified, the START and REPEAT operands cannot be specified.

"pathname"
pathname (see page 479) = name of the calendar file.

SYMDATE=
See CALENDAR.

sym-date-name
Symbolic date, which identifies the start time and any repetitions within the calendar file.

LIMIT=
determines the life of a calendar job. This limit applies in addition to the limits that are set by the calendar.

STD
The life of a calendar job is determined only from the entry of the symbolic date in the calendar.

number
This entry is only valid for calendar jobs.
1 ≤ number ≤ 32767
Maximum number of repetitions of the calendar job. Once a single job run has been completed, the run counter is incremented by 1. The system then checks whether the job counter has reached or exceeded the maximum number. If this occurs, the whole calendar job is terminated.

(DATE=yy-mm-dd,TIME=hh:mm)
The entry is only valid for calendar jobs.
Entries in the calendar file are only taken into account up to the specified limit. No further repetition job is generated for calendar entries after the limit; the calendar job terminates.
The limit refers exclusively to the scheduled jobs in the file, not to the real runtime of the jobs. Repetition jobs with a “permissible” start date are not subject to any further restrictions and are, for example, also started after the specified date if it was not possible to start them earlier because of delays in the job scheduler.
The date is determined by specifying the day and time:

See operand START=AT(...)

RUN-PRIO=
Defines the priority for the processing of the job (relative to other tasks).
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-LOGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value STD.

STD
Standard value for the selected job class. The default value is also accepted if invalid values are specified for “rprio”.

rprio
Specifies the run priority. MAXIMUM ≤ rprio ≤ 255. The lower the value, the higher the priority. The value for MAXIMUM is set in both the job class definition and the user catalog and can be accessed with the SHOW-JOB-CLASS or SHOW-USER-ATTRIBUTES command. If the values are not identical, the threshold value that is better for the user is accepted.

TIME=
Denotes the maximum CPU time (in seconds) that the task may use. The maximum amount of CPU time that can be specified is set by the selected job class.
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-LOGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value STD.

STD
Standard value of the selected job class.

t
CPU time in seconds. 0 ≤ t ≤ maximum CPU time.

NTL
No Time Limit. The task runs without a CPU time limit.

PROTECTION=
determines whether the job is protected from being terminated inadvertently by the CANCEL-JOB command.

NONE
The job is not protected.

CANCEL
The job is protected.

PRINT=
Denotes the maximum number of records that can be output by the task (in total) to the SYSLST, SYSLST01, SYSLST02,..., SYSLST99 system files. Data records that are output simultaneously to SYSOUT and SYSLST (LOG=(LISTING=YES) or MSG=FH specification) are not included.
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-LOGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value STD.

STD
Standard value of the selected job class.

number
Number of records. 0 ≤ number ≤ MAXIMUM. The value for MAXIMUM is set in the job class definition and can be accessed with the SHOW-JOB-CLASS command.

NO-LIMIT
The number of records is unlimited.

LOG=(...)
Specifies whether the log of the job run is also to be output to SYSLST (LISTING=YES).
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-OGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value *LISTING=NO.

JOB-PAR=
Enables the specification of additional attributes for the selected job class - provided they have been defined and disclosed by the system administration.
If the operand is not specified, when FROM-LOGON=YES it is assigned the value from the SET-OGON-PARAMETERS command in the ENTER file, otherwise it is assigned the value *NO.

***NO**
No additional attributes.

"attributes"
String of freely selectable characters; it is allocated by system administration for marking other job class attributes.

PRIORITY=
Designates the priority for the processing of a job (relative to other tasks).

p
Run priority. MAXIMUM ≤ p ≤ 255. The lower the value, the higher the priority. The value for MAXIMUM is set both in the job class definition and in the user catalog. It can be accessed with the SHOW-JOB-CLASS or SHOW-USER-ATTRIBUTES command. If the values are not identical, the threshold value that is better for the user is accepted.
Default value: standard value for the selected job class.

This standard value is also assumed if invalid values are specified for “p”.

([p],EXP[RESS])
The EXPRESS specification causes the ENTER job to be started immediately. This operand does not effect the further processing of the job. The authorization for the EXPRESS operand is set in the user catalog and/or in the job class definition.

The operand only continues to be supported for reasons of compatibility. The PRIORITY=p specification should be replaced by RUN-PRIO=rprio, and the PRIORITY=(p,EXPRESS) specification by RUN-PRIO=rprio, START=IMMEDIATELY.

The PRIORITY operand is ignored if the RUN-PRIO operand was specified; the EXPRESS specification is ignored if the START operand was specified.

MSG=
This operand is used to control the way in which system messages are to be output or job execution is to be logged.

F
The full system messages are output to the SYSOUT system file (F for “Full message”).

C
The coded short form of the system messages is output to SYSOUT (C for “Code”).

L
Console messages and operator responses for this job are logged (L for “Log”). If the user enters MSG=LH, the messages logged to SYSLST are accompanied by the time at which they were entered.

H
Execution is also logged to SYSLST (H for “Hold message”).

the MSG operand only continues to be supported for reasons of compatibility. MSG is completely ignored if LOG was specified.

addr
Address of the field to which the SYSOUT log is to be written. If omitted, or if the field has a length of zero, the log is output to SYSOUT only. The field must be aligned on a word boundary. Layout:

Bytes 0 - 1:	field length (≤ 32767 bytes)
Bytes 2 - 3:	no entry
Bytes 4 - n:	start of SYSOUT log.

The first 4 bytes of each record of the SYSOUT log transferred to the field contain the record length field (bytes 0-1 contain the record length, bytes 2-3 are reserved). The actual output text always starts at byte 4. Output records are written to the field until its boundary is reached. Any further output records that cannot be written to the field are output to SYSOUT only. If the field boundary is reached while a record is being written, the record may be truncated (error flag X'0C').

(r)
Register containing the “addr” address value.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Combining the START and REPEAT operands

START	REPEAT		
	AT-STREAM-STARTUP	DAILY bzw. WEEKLY	PERIOD
IMMEDIATELY / SOON	a)	c)	c)
AT / EARLIEST	a)	d)	f)
LATEST / WITHIN	a)	c)	g)
AT-STREAM-STARTUP	b)	e)	e)

a) The first job start and all subsequent job starts take place as specified.

b) The first job start takes place with START=AT-STREAM-STARTUP. All subsequent starts take place after the startup of the job scheduler with START=SOON.

c) The repetition cycle is based on the moment of job acceptance.

d) The repetition cycle is based on the specified time (START=...., TIME=....).

e) The first job start takes place after the startup of the job scheduler. The repetition cycle is based on this start time. Subsequent starts take place with START = SOON.

f) The repetition cycle is based on the specified time (START=...., TIME=....). The second start and all subsequent starts take place with START=SOON.

g) The repetition cycle is based on the moment of job acceptance. All subsequent starts take place with START=SOON.

General notes

- The operand list of the command operands must be enclosed in single quotes.
- A copy of the file is created using the name S.IN.tsn.date.hhmmss in the following cases:
  - if the file resides on private disk
  - if the ENTER job is to run under a different user ID than that under which the file is cataloged
  - if the file is a temporary file
  - if the file is encrypted

  If the ENTER file is a library element, a copy is created under the name S.IN.libraryname.elementname.tsn.hhmmss.
  The S.IN. file is automatically deleted at the end of the job (EXIT-JOB) unless checkpoints were set during job execution (**WRCPT** macro). In this case, the S.IN. file must be present to ensure a restart without problems (RESTART-PROGRAM command).
- Although S.IN. files are password protected (EXEC-PASSWORD) it is possible to erase them with /DELETE-FILE without first entering the password. It is thus possible to remove any S.IN. files that are no longer required or that have not been erased by the system.
- ENTER files can be password protected against being read (READ-PASSWORD), overwritten (WRITE-PASSWORD) or executed (EXEC-PASSWORD) (CREATE-FILE command). The EXEC-PASSWORD or a higher-ranking password must be specified in the FPASS operand before an ENTER-JOB command is issued. WRITE-PASSWORD must also be specified if the file is to be erased after execution (ERASE=YES operand). The passwords are checked for errors as soon as the ENTER call is processed.
  A successful check remains valid even if the user subsequently changes the passwords, and the file is executed.
- ENTER files may be SAM or ISAM files with variable record length (RECFORM=V). 72 characters are interpreted for each data record. Where ISAM files are concerned, the key field may be at any position in the data record, since it is masked out.
  See “Consistent job interface” under the ENTER-JOB command.

Notes on job monitoring (see the “JV” manual [22])

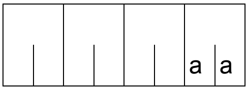
- At ENTER time, the status indicator of “jvname” is set to “$S”, the “TSN” indicator to the job number associated with the job and the processor indicator to the catalog ID of the processor which is executing the job.
- If `jvname` cannot be accessed at macro processing time, the call is rejected. If the job variable can only not be accessed later (the ENTER job wishes to enter a value), the job outputs an error message to SYSOUT and continues to run normally.
- The user ID under which the monitoring job variable is issued and also the user ID for which the job is processed must have access to `jvname`.
- JVPASS is the password, in accordance with the password hierarchy, which allows access to the monitoring job variable. The password must be specified in the **ENTER** macro if job distribution is required (see the “HIPLEX MSCF” manual [26]). Without job distribution, the password may be given via a separate ADD-PASSWORD command.
- For access to the monitoring job variable, the same rules apply as for access to the ENTER file.

Notes on job distribution (see the “HIPLEX MSCF” manual [26])

- “hostid” must define an active system of the MSCF network, otherwise the **ENTER** call will be rejected.
- “jvname1” must contain the “hostid” of an active system of the MSCF network, otherwise the **ENTER** call will be rejected.
- “catid” must define a known and accessible catalog (within the MSCF network), otherwise the **ENTER** call will be rejected.
- “jvname2” must contain the “catid” of a known and accessible catalog (within the MSCF network), otherwise the **ENTER** call will be rejected.
- If the HOST and CAT operands are specified, the value of the HOST operand is used to define the target system.
- All passwords (both FPASS and CRPASS for the ENTER file and JVPASS for the MONJV) must be specified in the **ENTER** macro when job distribution is required. Without job distribution, the password can also be given using a separate ADD-(CRYPTO-)PASSWORD command.

Return information and error flags

R15:



A return code relating to the execution of the ENTER macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Normal termination.
X'04'	The request has not been processed due to insufficient memory area.
X'08'	Error in the operand list (address area).
X'0C'	The last output record entered in the user area has been truncated.
X'10'	Macro call/command error (the command returned an error to MCLP), e.g. file not cataloged (DMS0D33) or incorrect operand (JMS0021).

ETABIT – Generate or change entry for symbol table

General

Application area:	Linking and loading; see page 47
Macro type:	Type S, MF format **2**: standard/C/D/L/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **ETABIT** macro generates an entry for a symbol table, which is transferred to the DBL when the **ETABLE** macro is called.

Macro format and description of operands

ETABIT
MF=D / C / L / M
,AMODE=*NOT-SPECIFIED / *31 / *24 / *ANY
,HSI_CODE=*BY-SYSTEM / *390 / *RISC / *SPARC / *X86E [1]
,INVISIBLE=*NOT-SPECIFIED / *NO / *YES
,LEN=0 / <integer 0..2147473647>
,LOAD_ADDR=NULL-1 / <var: pointer>
,PAGE_ALIGNED=*NOT-SPECIFIED / *NO / *YES
,PRIVILEGED=*NOT-SPECIFIED / *NO / *YES
,PUBLIC=*NOT-SPECIFIED / *NO / *YES
,READ_ONLY=*NOT-SPECIFIED / *NO / *YES
,RESIDENT=*NOT-SPECIFIED / *NO / *YES
,SYMBOL_NAME='␣' / <c-string 1..32> / <var: char 1..32>
,SYMBOL_TYPE=*NOT-SPECIFIED / *CSECT / *ENTRY / *COMMON
,PARAM=<var: pointer> / (reg: pointer>)
,PREFIX=P / p
,MACID=BET / macid

[1] The operand values *RISC and *SPARC have no meaning in BS2000/OSD-BC V9.0 and higher

When creating an entry, the value *NOT-SPECIFIED causes the attributes of the symbols to be initialized with ’FALSE’.

The operands are described in alphabetical order below.

AMODE=*NOT-SPECIFIED / *31 / *24 / *ANY
Value of the AMODE attribute

HSI_CODE=*BY-SYSTEM / *390 / X86E
The code type that refers to the symbol must be specified (*390, *X86E).
The default value is the type of server on which the user program is running. However, the correct HSI code should always be entered in order to avoid an undesired and possibly incorrect update on an `ETABLE ACTION=*UPDATE` macro.

INVISIBLE=*NOT-SPECIFIED / *NO / *YES
Value of the INVISIBLE attribute

LEN=0 / <integer 0..2147483647>
Length of element.
For CSECTs and COMMON areas, LEN must be greater than 0.
For ENTRYs, LEN must be 0.

LOAD_ADDR=NULL-1 / <var: pointer>
Load address of the symbol.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM, PREFIX and MACID, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
It is possible to specify a PREFIX in the C form, D form, or M form of the macro, and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

PAGE_ALIGNED=*NOT-SPECIFIED / *NO / *YES
Value of the PAGE_ALIGNED attribute

PRIVILEGED=*NOT-SPECIFIED / *NO / *YES
Value of the PRIVILEGED attribute

PUBLIC=*NOT-SPECIFIED / *NO / *YES
Value of the PUBLIC attribute

READ_ONLY=*NOT-SPECIFIED / *NO / *YES
Value of the READ_ONLY attribute

RESIDENT=*NOT-SPECIFIED / *NO / *YES
Value of the RESIDENT attribute

SYMBOL_NAME='␣' / <c-string 1..32> / <var: char 1..32>
Symbol name

SYMBOL_TYPE=*NOT-SPECIFIED / *CSECT / *ENTRY / *COMMON
Symbol type entered in the symbol table.

Notes on the macro call

- If the HSI_CODE parameter is not specified, it is set to the default value *BY-SYSTEM. A subsequent call to the **ETABLE** macro with ACTION=*UPDATE could, in some cases, cause the HSI code to be inadvertently updated. To prevent this from happening, the HSI code of the symbol must always be specified.
- The notes relating to the **ETABLE** macro also apply to the **ETABIT** macro. The layout of a table entry is also described there.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the ETABIT macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally.
X'01'	X'00'	X'0000'	The same ETABLE symbol already exists.
X'60'	X'01'	X'0006'	Error in addressing mode.
X'60'	X'01'	X'0007'	Invalid length specification.
X'60'	X'01'	X'0009'	Invalid attribute value.
X'60'	X'01'	X'000B'	Invalid value in HSI.
X'60'	X'01'	X'000C'	The symbol table entry for ETABLE is invalid.
X'60'	X'01'	X'000D'	SYMBOL_TYPE was not specified when the entry was generated.
X'60'	X'01'	X'002C'	Invalid symbol name.
X'60'	X'01'	X'0060'	Symbol not found.
X'60'	X'01'	X'0130'	Invalid LOAD_ADDR operand.
X'60'	X'01'	X'0150'	Error in symbol type.
X'60'	X'01'	X'0151'	A symbol generated with ETABLE with this name already exists.
X' 60'	X' 01'	X'0152'	A symbol not generated with ETABLE with this name already exists and the required action for such a symbol is not permitted (see page 503).
X' 00'	X' 01'	X'FFFF'	The function is no longer or not yet supported.
X' 00'	X' 03'	X'FFFF'	The interface version is not supported.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

ETABLE – Transfer load information

General

Application area: Linking and loading; see page 47
Macro type: Type S, MF format **2**: standard/C/D/L/E/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

With the **ETABLE** macro, the user program transfers a symbol table to the DBL; the table is integrated into the symbol table of the specified context. The transferred table notifies the DBL of the names and attributes of CSECTs, ENTRYs, and COMMON areas of the user program.

This macro is not compatible with the previous TABLE macro.

Macro format and description of operands

ETABLE
MF=S / D / C / E / L / M ,ACTION=*CREATE / *UPDATE / *DELETE ,CONTEXT_NAME=' ␣' / <char 1..32> ,CONTEXT_STATE=*DBL-OPTIONS / *ANY / *NEW / *OLD ,TABLE_ADDRESS=NULL-1 / <var: pointer> ,TABLE_LENGTH=0 / <integer 0..2147483647> ,PARAM=<var: pointer> / (reg: pointer>) ,PREFIX=P / p ,MACID=BEI / macid

The operands are described in alphabetical order below.

ACTION=
Action which is to be executed for the individual symbols in the transferred table.

***CREATE**
The symbols are entered into the symbol table of the context. They must not already be present in it.

***UPDATE**
The symbols are entered with their new attributes into the symbol table of the context. They must already be present in it. The visibility of symbols which were not entered with an earlier call of the **ETABLE** macro is also changed here.

***DELETE**
The symbols are to be deleted from the symbol table of the context.

CONTEXT_NAME='␣' / <char 1..32>
Name of the context to which the symbols are assigned. The first character must be a letter.

CONTEXT_STATE=
Status of the context specified with CONTEXT_NAME

***DBL-OPTIONS**
The operand value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the operand has not yet been set using the MODIFY-DBL-DEFAULTS command, the value that follows *DBL-OPTIONS in the syntax definition applies.

***ANY**
If the context already exists it is used, otherwise a new context is created.

***NEW**
The context is created. It must not already exist.

***OLD**
The context already exists.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM, PREFIX and MACID, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
It is possible to specify a PREFIX in the C form, D form, or M form of the macro, and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

TABLE_ADDRESS=NULL-1 / <var: pointer>
Address of a table of symbols to be transferred to DBL.

TABLE_LENGTH=0 / <integer 0..2147483647>
Actual length of the table (in bytes)

Notes on the macro call

- The type and name of a symbol cannot be changed with **ETABLE**.
- The address in the LOAD_ADDR field must be a valid class 6 memory address.
- A symbol that was entered in the symbol table of the context with **ETABLE** cannot be deleted from the symbol table using the UNBIND macro. However, symbols that were introduced with the LOAD-/START-EXECUTABLE-PROGRAM (or LOAD-/START-PROGRAM) command or with the BIND and TABLE macros cannot be deleted with **ETABLE** ACTION=*DELETE.
- If ACTION=*DELETE is set, the CONTEXT_STATE operand is ignored. Only the symbol names are taken into account.
- The user is responsible for symbols entered in the symbol table with **ETABLE**; these symbols are not visible for the current task.
- If a COMMON area is introduced with **ETABLE**, DBL assumes that the COMMON area is already loaded and refers the LOAD_ADDR field to an allocated memory area.
- If ACTION=UPDATE is applied to a symbol which was not generated by ETABLE, only the symbol name, symbol type and the INVISIBLE attribute are taken into account during processing.
- Only the visibility can be updated for a symbol which was not generated by ETABLE.
- As the visibility can also be updated for symbols which were not generated by ETABLE, the visibility of the first symbol found is always updated.

Symbol table format

The address specified with TABLE_ADDRESS points to a symbol table, which may contain a number of entries. An individual entry has the following format:

Byte	Length	Field name	Meaning and/or value
0	8	HDR	Standard header
8	1	TYPE	/ CSECT (X'F0') or / ENTRY (X'F1') or / COMMON (X'F3')
9	7	ATTRIBUTE	Same attributes as for CSECTs (1 byte per attribut with the following meaning): 1. INVISIBLE: X'01' ≙ NO, X'02' ≙ YES 2. AMODE: X'01' ≙ 31, X'02' ≙ 24, X'03' ≙ ANY 3. RESIDENT: X'01' ≙ NO, X'02' ≙ YES 4. PAGE_ALIGNED: X'01' ≙ NO, X'02' ≙ YES 5. READ_ONLY: X'01' ≙ NO, X'02' ≙ YES 6. PUBLIC: X'01' ≙ NO, X'02' ≙ YES 7. PRIVILEGED: X'01' ≙ NO, X'02' ≙ YES
16	4	LOAD_ADDR	Load address of the symbol
20	2	LEN	Length of the CSECT / of the COMMON area
22	32	SYMBOL_NAME	Name of the symbol
54	1	HSI_CODE	390 (X'01') or x86E (X'09')
55	1	MMODE	TU_4K_DEPENDENT (X'02') or COMPATIBLE (X'03) or NATIVE (X'04')

The DSECT for such a symbol entry is generated with `ETABIT MF=D`.

Handling of name conflicts

The following table shows how name conflicts between symbols introduced by the **ETABLE**, **TABLE** and **BIND** macros are handled:

	Existing symbol was introduced by		
ACTION	ETABLE	TABLE	BIND
*CREATE	(1)	(2)	(2)
*UPDATE	(3)	(4)	(4)
*DELETE	(5)	(6)	(6)

(1) An existing **ETABLE** symbol has the same name.
The following return code is transferred: ETABLE_SYMB_DUPLICATE

(2) A name conflict occurs. The following return code is transferred:
ETABLE_NAME_COLLISION

(3) The symbol is changed (no name conflict).

(4) Only the visibility of the symbol is updated.

(5) The symbol is deleted (no name conflict).

(6) Invalid action.

Return information and error flags

If an error occurs during processing of the table entries, a corresponding return code is entered in the HDR field of the table entry containing the error. The return code FUNCTION_PARTIALLY_PROCESSED is entered in the standard header of the parameter list and processing of the table entries is continued.

The PROCESSED_ITEMS field of the parameter list contains the number of correctly processed entries.

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the ETABLE macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally.
X'01'	X'00'	X'0000'	The function was already executed.
X'02'	X'00'	X'0001'	The function was partially executed.
X'60'	X'01'	X'0001'	Invalid TABLE_LENGTH operand.
X'60'	X'01'	X'0002'	Invalid specification for TABLE_ADDRESS or TABLE_LENGTH.
X'60'	X'01'	X'0008'	Invalid ACTION operand
X'60'	X'01'	X'000A'	Internal ETABLE error.
X'60'	X'01'	X'0019'	A reserved field does not contain any binary zeros.
X'60'	X'01'	X'0040'	The context is not yet present and CONTEXT_STATE=*OLD was specified.
X'60'	X'01'	X'0048'	The context is already present and CONTEXT_STATE=*NEW was specified.
X'60'	X'01'	X'0134'	Invalid CONTEXT_STATE parameter
X'60'	X'01'	X'0148'	Invalid context name
X'60'	X'40'	X'0158'	Maximum number of user contexts has been reached.
X'60'	X'20'	X'0200'	Internal DBL error
X'60'	X'20'	X'0300'	System error
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported.
X'00'	X'03'	X'FFFF'	The interface version is not supported.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Example

This example is designed to illustrate the application of the ETABLE/ETABIT interfaces:

```
* DSECT of ETABLE item
*
         ETABIT MF=D
         ...
* Initialization of the ETABLE parameter list
*
         MVC   ETAPL(PBEI#),ETAPLI
         LA    1,ETATAB
         LA    2,ETATABL
         ETABLE MF=M,TABLE_ADDRESS=(1),TABLE_LENGTH=(2)
*
* Initialization of the entries
*
         USING PBETDS,1                reg 1 = Address of the first entry
*                                      PBETDS = DSECT for the entry
*
* First entry
*
         MVC   0(PBET#,1),ETAITEMI                Initializing entry
         ETABIT MF=M,SYMBOL_NAME=CS1NAM,LEN=CS1L,
               LOAD_ADDR=CS1@,SYMBOL_TYPE=*COMMON
*
* Second entry
*
         LA    1,PBET#(1)              reg 1 = Address of the second entry
         L     2,CS2@
         MVC   0(PBET#,1),ETAITEMI               Initializing entry
         ETABIT MF=M,SYMBOL_NAME='CS2',LEN=200,
               LOAD_ADDR=(2),SYMBOL_TYPE=*CSECT
*
* Call ETABLE
*
         ETABLE MF=E,PARAM=ETAPL
*
* Return code evaluation
*
         ...
*
* ETABLE table
*
ETATAB   DS    XL(2*PBET#)    Table for two entries
ETATABL  EQU   *-ETATAB       Table length
         DS    0F
CS1@     DS    F              Address of the first symbol entered
```

```
CS1NAM   DS    CL32           Name of the first symbol entered
CS1L     DS    Y              Length of the first symbol entered
CS2@     DS    F              Address of the second symbol entered
*
ETAPL    ETABLE MF=C          Parameter list
*
* Structure for initializing the ETABLE parameter list
*
ETAPLI   ETABLE MF=L,ACTION=*CREATE,CONTEXT_STATE=*NEW,
              CONTEXT_NAME='ETACTX'
*
* Structure for initializing an ETABLE entry
*
 ETAITEMI ETABIT MF=L
```

EXIT – Terminate STXIT process/routine

General

Application areas:	Starting, interrupting and terminating; see page 72 STXIT processing; see page 131
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

Program interrupts can be processed with STXIT routines. They run as separate processes (and have their own PCB and processing level).
No return code relating to the execution of the **EXIT** macro is transferred.

Macro description

The **EXIT** macro terminates an STXIT process.

The user may specify whether another STXIT routine assigned to the same event class is to be activated; if this is not the case, the interrupted process is continued or the program is terminated.

Notes

- If the STXIT routine is assigned to the “ABEND” or “normal program termination” event class, the program is terminated by the system after the **EXIT** macro, provided that all the STXIT routines for these STXIT event classes have been executed (if CONTINU=YES) and the event (e.g. LOGOFF(ABEND)) has been terminated by the system.
- An STXIT routine assigned to the “end of the program runtime” event class should be terminated with the **TERM** macro.

Macro format and description of operands

```
EXIT

CONTINU=YES / NO / STD

        ⎧ NO                        ⎫
,TERM=  ⎨ (PRGR[,DUMP]   ,NORMAL])  ⎬
        ⎩ (STEP[,DUMP] [,NORMAL])   ⎭
[,MF=L / (E,..)]
```

CONTINU=

Defines whether another STXIT routine of the same event class is to be started.

YES

Another STXIT routine is started.

NO

No other STXIT routine is started.

STD

This option normally corresponds to CONTINU=YES. It has a different meaning only if all of the following three conditions are fulfilled:

- the STXIT class is 'PROGRAM CHECK' or 'UNRECOVERABLE PROGRAM ERROR',
- all previous STXIT routines of this STXIT class have terminated with CONTINU=STD, and
- there are no more STXIT routines of the same STXIT class.

In this case the program is resumed as if these STXIT routines had never existed, i.e. the error recovery envisaged for the fault which has occurred is started.

TERM=

Specifies whether the program is to be terminated (if no other STXIT routines of the same STXIT event class are activated in the program).

NO

The program is not to be terminated.

(PRGR[,DUMP][,NORMAL])

The program is to be terminated.
If DUMP is specified, a user dump is also output.
If NORMAL is specified, the program is terminated normally.
Default setting is ABNORMAL, i.e. the program is terminated abnormally.

(STEP[,DUMP][,NORMAL])

In an interactive task, the program is terminated.
In a batch task, the program additionally branches to the next SET-JOB-STEP, or EXIT-JOB command.
If DUMP is specified, a user dump is also output. If NORMAL is specified, the program is terminated normally.
Default setting is ABNORMAL, i.e. the program is terminated abnormally.)

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

For an **example**, see section “STXIT procedure with contingency processing” on page 131.

GCCSN – Display CCS name for command and data input

General

Application areas:	Input/output of files and records; see page 156 Requesting and accessing lists and tables; see page 155
Macro type:	Type S, MF format **3**: D/C/S/E/L form; see page 29

Macro description

The **GCCSN** macro allows the user to display the name of the current coded character set (coding table) for input and output of commands or data.
The name of the current coded character set (coding table) depends on the input or output source:

– SYSDTA/SYSCMD/SYSOUT is assigned to a data display terminal:
The CCS name is determined by VTSU. The **GCCSN** macro receives the name (internally) from the ACTCH field of the **TSTAT** macro's parameter list.

– SYSDTA/SYSCMD/SYSOUT/SYSLST is assigned to an S variable:
Interactive mode: the same as when SYSDTA/SYSCMD is assigned to a data display terminal.
Batch mode: the CCS name 'EDF03IRV' is displayed.

– SYSDTA/SYSCMD/SYSOUT/SYSLST is assigned to a PLAM library element:
The CCS name of the library element is displayed.
If no coding table is assigned to the library element, the coding table name “EDF03IRV” is displayed.

– SYSCMD is assigned to a file:
The CCS name in the catalog entry for the file is displayed. Files may be cataloged files, S procedures and non-S procedures.

– SYSDTA/SYSOUT/SYSLST is assigned to a file:
The CCS name in the catalog entry for the file is displayed. Files may be cataloged files, S procedures and non-S procedures (if SYSDTA=(SYSCMD)).
The name of the coding table for SYSOUT/SYSLST can be assigned using the commands `/ASSIGN-SYSOUT/SYSLST CODED-CHARACTER-SET=`.
If no coding table is assigned, the default name of the user catalog entry is displayed.

Once the parameter list has been initialized (MF=L) the function call (MF=E) is issued. If the macro is executed without errors, the field `<PREFIX><MACID>CCSN` in the parameter list contains the CCS name of the queried system file.

Macro format and description of operands

GCCSN
STREAM=SYSDTA / SYSCMD / SYSOUT / SYSLST ,MF=D / C / S / E / L [,PARAM=addr / (r)] ,PREFIX=C / p ,MACID=CSN / macid

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form or D form of the macro and additionally a MACID in the C form (see page 29).

It is also possible to call this macro with MF=S.

STREAM=
Specifies the system file whose CCS name is to be output.

SYSDTA
The CCS name of SYSDTA is to be displayed.

SYSCMD
The CCS name of SYSCMD is to be displayed.

SYSOUT
The CCS name of SYSSYSOUT is to be displayed.

SYSLST
The CCS name of SYSLST is to be displayed.

Notes

- If SYSDTA is assigned to a data display terminal, the user can modify the CCS name dynamically in a program (in the VTSU control block of the **RDATA** macro).
  This modification is valid only during this **RDATA** input/output and has no effect on the output of the **GCCSN** macro, which in this case always displays the CCS name that is predefined in the user's user catalog.
- If SYSCMD is assigned to a data display terminal, *privileged* users can modify the CCS name dynamically (in the VTSU control block of the **WRTRD** macro).

This modification is valid only during this **WRTRD** input/output and has no effect on the output of the **GCCSN** macro, which in this case always displays the CCS name that is predefined in the user's user catalog.

- If SYSDTA/SYSCMD/SYSOUT/SYSLST is assigned to a file or a library element, the CCS name does not change between the time the file or library element is opened (OPEN) and the time it is closed (CLOSE).
- If a procedure is interrupted with K2, the current SYSDTA CCS name is not that of the data display terminal but that of the procedure that has just been interrupted.
- If, during processing of a procedure, a procedure parameter is prompted at the data display terminal, the current CCS name for this procedure applies to the fixed (requesting) part of the command/statement line. The CCS name of the data display terminal applies to the parameter value entered.

  Example

  ```
  Level 1:  data display terminal   Level 2:  procedure HALLO

  :
  (IN)  /CALL-PROC HALLO ----->     /BEGIN-PROC A,PROC-PAR=(&PARAM),-
                                    /   ESC-CHAR='&'
                                    /ASS-SYSDTA *SYSCMD
                                    /START-PROG $EDT
   <----------------------------     @READ'&PARAM
  (OUT) &PARAM=
  (IN)  TEST.1
   ---------------------------->     @READ'TEST.1
                                     :
  ```

  The string `@READ'` is read using the CCS name for the HALLO procedure; the string `TEST.1` is read using the CCS name for the data display terminal.

- The user can issue the **RDATA ..,A** macro to obtain information on any modifications to the SYSDTA assignment.

Layout of the CSECT

```
           GCCSN MF=C
1          #INTF REFTYPE=REQUEST,INTNAME=GCCSN,INTCOMP=002
1          MFCHK MF=C,SUPPORT=(C,D,S,L,E),PREFIX=C,                      C
1                MACID=CSN,DMACID=CSN,                                   C
1                PARAM=,ALIGN=F,SVC=39
2          DS    0F
2                *,##### PREFIX=C, MACID=CSN #####
1 CCSNCS   FHDR  MF=(C,CCSN)
2 CCSNCS   DS    0A
2 CCSNFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 CCSNIFID DS    0A              0   INTERFACE IDENTIFIER
2 CCSNFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 CCSNFCT  DS    AL1             2   FUNCTION NUMBER
2 CCSNFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 CCSNRET  DS    0A              4   GENERAL RETURN CODE
2 *
2 * GENERAL_RETURN_CODE CLEARED (X'00000000') MEANS
2 * REQUEST SUCCESSFUL PROCESSED AND NO ADDITIONAL INFORMATION
2 *
2 CCSNSRET DS    0AL2            4   SUB RETURN CODE
2 CCSNSR2  DS    AL1             4   SUB RETURN CODE 2
2 * ALWAYS CLEARED (X'00') IF MAIN_RETURN_CODE IS X'FFFF'
2 * Standard subcode2 values as defined by convention:
2 CCSNR2OK EQU   X'00'               All correct, no additional info
2 CCSNR2NA EQU   X'01'               Successful, no action was necessary
2 CCSNR2WA EQU   X'02'               Warning, particular situation
2 CCSNSR1  DS    AL1             5   SUB RETURN CODE 1
2 *
2 * GENERAL INDICATION OF ERROR CLASSES
2 *
2 * CLASS A    X'00'           FUNCTION WAS SUCCESSFULLY PROCESSED
2 * CLASS B    X'01' - X'1F'   PARAMETER SYNTAX ERROR
2 * CLASS C    X'20'           INTERNAL ERROR IN CALLED FUNCTION
2 * CLASS D    X'40' - X'7F'   NO CLASS SPECIFIC REACTION POSSIBLE
2 * CLASS E    X'80' - X'82'   WAIT AND RETRY
2 *
2 CCSNRFSP EQU   X'00'               FUNCTION SUCCESSFULLY PROCESSED
2 CCSNRPER EQU   X'01'               PARAMETER SYNTAX ERROR
2 *  3 GLOBALLY DEFINED ISL ERROR CODES IN CLASS X'01' - X'1F'
2 CCSNRFNS EQU   X'01'               CALLED FUNCTION NOT SUPPORTED
```

```
2 CCSNRFNA EQU   X'02'                 CALLED FUNCTION NOT AVAILABLE
2 CCSNRVNA EQU   X'03'                 INTERFACE VERSION NOT SUPPORTED
2 *
2 CCSNRAER EQU   X'04'                 ALIGNMENT ERROR
2 CCSNRIER EQU   X'20'                 INTERNAL ERROR
2 CCSNRCAR EQU   X'40'                 CORRECT AND RETRY
2 *  2 GLOBALLY DEFINED ISL ERROR CODES IN CLASS X'40' - X'7F'
2 CCSNRECR EQU   X'41'                 SUBSYSTEM (SS) MUST BE CREATED
2 *                                    EXPLICITELY BY CREATE-SS
2 CCSNRECN EQU   X'42'                 SS MUST BE EXPLICITELY CONNECTED
2 *
2 CCSNRWAR EQU   X'80'                 WAIT FOR A SHORT TIME AND RETRY
2 CCSNRWLR EQU   X'81'                      "     LONG       "
2 CCSNRWUR EQU   X'82'                 WAIT TIME IS UNCALCULABLY LONG
2 *                                    BUT RETRY IS POSSIBLE
2 *  2 GLOBALLY DEFINED ISL ERROR CODES IN CLASS X'80' - X'82'
2 CCSNRTNA EQU   X'81'                 SS TEMPORARILY NOT AVAILABLE
2 CCSNRDH  EQU   X'82'                 SS IN DELETE / HOLD
2 *
2 CCSNMRET DS    0AL2             6    MAIN RETURN CODE
2 CCSNMR2  DS    AL1              6    MAIN RETURN CODE 2
2 CCSNMR1  DS    AL1              7    MAIN RETURN CODE 1
2 *
2 * SPECIAL LAYOUT OF LINKAGE_MAIN_RETURN_CODE (YYYY IN X'00XXYYYY')
2 *
2 CCSNRLNK EQU   X'FFFF'               LINKAGE ERROR / REQ. NOT PROCESSED
2 CCSNFHL  EQU   8                8    GENERAL OPERAND LIST HEADER LENGTH
2 *
1 CCSNFLAG   DS    X                   STREAM IDENTIFIER
1 CCSNDTA    EQU   X'01'               SYSDTA
1 CCSNCMD    EQU   X'02'               SYSCMD
1 CCSNOUT    EQU   X'03'               SYSOUT
1 CCSNLST    EQU   X'04'               SYSLST
1 CCSNRES1   DS    CL3                 RESERVED
1 CCSNRES2   DS    A                   RESERVED
1 CCSNRES3   DS    A                   RESERVED
1 CCSNCCSN   DS    CL8                 CODED CHARACTER SET NAME
1 CCSNPAR    EQU   X'01'               SC1 RC : PARAMETER ERROR
1 CCSNERR    EQU   X'20'               SC1 RC : INTERNAL ERROR
1 CCSN#      EQU   *-CCSNCS            PARAMETER LIST LENGTH
1          SPACE 2
```

Return information and error flags

During macro processing, register R1 receives the parameter list address. The CCS name is transferred in the field `<PREFIX><MACID>CCSN` in the parameter list.

Standard header:

0	0	b	b	a	a	a	a

A structured return code relating to the execution of the GCCSN macro is transferred (bb=Subcode1, aaaa=Maincode):

X'bb'	X'aaaa'	Meaning
X'00'	X'0000'	Normal execution.
X'01'	X'0008'	Operand error.
X'20'	X'0004'	Internal error.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

GEPRT – Get program time

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Type O/type R; see page 28 Type S, MF format **1**: 31-bit interface standard/E/L/D form; see page 29

- O-type macro expansion does not make use of any registers.
- The SVCs used differ, depending on the macro form.

Macro description

The **GEPRT** macro provides information on the CPU time used by the job since SET-LOGON-PARAMETERS and/or the CPU time still available for the executing program. The CPU time still available is determined from the time limit allowed for execution of the program (as specified in the START-PROGRAM command, CPU-LIMIT operand); or, if this operand was omitted, from the time limit for the job (i.e. either the specified time or the default value in the SET-LOGON-PARAMETERS command).
Fields must be created in order to receive the program time data (provided in zoned decimal format).

Calling the macro without specifying any operands (type R) will provide the CPU time used since SET-LOGON-PARAMETERS. This information (in TODR format) is passed to the user in registers R0 and R1. Without operands, the function executes considerably faster than when calling the macro with operands (only 220 instructions for type R, as opposed to 550 for type S or O).

Macro format and description of operands

```
GEPRT
--------------------------------------------

[ { [addr1] [,addr2] } [,FORMAT=B8]]
  { [(r1)] [,(r2)]   }

[,PARMOD=31]

[,MF=L / (E,..) / D]

,PREF=G / p
```

addr1
Symbolic address of a 6-byte field. The CPU time already used by the job is entered in this field.
Field length = 8 bytes if the operand FORMAT=B8 is specified.

(r1)
Register containing the address value “addr1”. If PARMOD=31, the address value may not be transferred in register R1.

addr2
Symbolic address of a 6-byte field. The remaining (maximum) CPU time for the program or the job is entered into this field.
Field length = 8 bytes if the operand FORMAT=B8 is specified.

(r2)
Register containing the address value “addr2”. If PARMOD=31, the address value may not be transferred in register R1.

FORMAT=B8
The time information is provided in the format hhhhmmss (hhhh=hours, mm=minutes, ss=seconds).

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A prefix PREF (1 letter, default setting: G) can be specified in the D form of the macro (see page 29).

PARMOD=
Controls macro expansion. If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default value for the assembler (= 24-bit interface).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Functional description

The CPU time is supplied in zoned decimal format as a 6-digit string (hhmmss) or, if FORMAT=B8 is specified, as an 8-digit string (hhhhmmss).

Specifications for jobs with no CPU time limit (NTL):

Field for elapsed CPU time:

	current value < 100h	current value ≥ 100h
6-byte field 8-byte field	current value current value	99h 59min 59sec current value

Field for remaining CPU time:

6-byte field	99h 59min 59sec
8-byte field	9999h 59min 59sec

Note

The **GEPRT** macro transfers no return code. Addressing errors made by the user cause the system to generate the “address error” event (STXIT event class “unrecoverable program error”). The program is terminated with the flag “address error” unless an STXIT routine has been included in the program to deal with this event.

Example

```
GEPRT    START
         PRINT NOGEN
         BALR  3,0
         USING *,3
         GEPRT FIELD1,FIELD2,FORMAT=B8 ——————————————————————————————————  (1)
         WROUT OUTB,ERROR
ERROR    TERM
****     DEFINITIONS     *******
OUTB     DC    Y(OUTBE-OUTB)
         DS    CL3
         DC    C'CPU TIME USED: '
FIELD1   DS    CL8
         DC    C'  AVAILABLE CPU TIME: '
FIELD2   DS    CL8
OUTBE    EQU   *
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,geprt), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,geprt))
%  ASS6011 ASSEMBLY TIME: 337 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 80 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=geprt
%  BLS0523 ELEMENT 'GEPRT', VERSION '@' FROM LIBRARY
   ':20SG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'GEPRT', VERSION ' ' OF '<date> <time>' LOADED
CPU TIME USED: 00000059  AVAILABLE CPU TIME: 00022901  ————————————————  (2)
```

(1) Both the CPU time used since SET-LOGON-PARAMETERS and the CPU time still available for running the program are requested.

(2) The CPU time used since SET-LOGON-PARAMETERS and the CPU time still available for running the program are output.

GETPRGV – Get program version

General

Application area:	Linking and loading; see page 47
Macro type:	Type S, MF format **2**: standard/C/D/E/L/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **GETPRGV** macro outputs the program version that was previously selected by the user with the **SELPRGV** macro or the SELECT-PROGRAM-VERSION command.

Macro format and description of operands

GETPRGV
MF=S / C / D / E / L / M ,PRGNAME=<name 1..32> ,PRGNAM@=<var: name 32..32> / (<reg: pointer>) ,PRGVER@=<var: structure> / <var: pointer> / (<reg: pointer>) ,PARAM=<var: pointer> / (<reg: pointer>) ,PREFIX=P / p ,MACID=BGT / macid

PRGNAME=<name 1..32>
Program name. The name may contain alphanumerical characters only.
May be specified only if MF=L or MF=S.

PRGNAM@=<var: name 32..32> / (<reg: pointer>)
Symbolic address or register containing the address of a 32 character field, which contains the the program name. Shorter name specifications must be padded with blanks.
May be specified only if MF=M.

PRGVER@=
Address of a structure in which the program was entered by DBL if a version was selected. The DSECT <prefix><macid>VRDS for the structure is generated if MF=D and has the following layout:

Byte	Length	Field name	Description
0	1	VERL	Program version length
1	24	VERS	Program version

<var: structure>
Symbolic address of the structure. May be specified only if MF=L or MF=S.

<var: pointer>
Symbolic address of an auxiliary field containing the address of the structure.
May be specified only if MF=M.

(<reg: pointer>)
Register containing the address of the structure. May be specified only if MF=M.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM, PREFIX and MACID, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

It is possible to specify a PREFIX in the C form, D form, or M form of the macro, and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

Notes on the macro call

- DBL first searches the program versions for which SCOPE=PROGRAM was set (see **SELPRGV** macro). If DBL finds the specified program among them it transfers the program version and terminates the search.
- Valid class 6 memory addresses must be specified in PRGNAM@ and PRGVER@.

Return information and error flags

The program version and its length are transferred in the structure that was specified using the PRGVER@ parameter.

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the GETPRGV macro is transferred in the standard header
(cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	**X'bb'**	**X'aaaa'**	**Meaning**
X'00'	X'00'	X'0000'	The macro was executed normally.
X'00'	X'01'	X'0001'	PRGNAME not specified or invalid.
X'00'	X'01'	X'0002'	PRGVER@ not specified or invalid.
X'00'	X'00'	X'0004'	A version for the program has not been selected.
X'00'	X'00'	X'0007'	Program not found with the name specified.
X'00'	X'20'	X'0300'	System error.
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported.
X'00'	X'03'	X'FFFF'	The interface version is not supported.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

GPARMOD – Control macro expansion

General

Application area:	XS programming; see page 164
Macro type:	Type O; see page 28

- the user has the choice between a 24-bit addressing mode and a 31-bit addressing mode (AMODE=24/31). Programs stored above the 16-Mb boundary must be run in 31-bit addressing mode.
- Those macros which did not have 31-bit interfaces have been extended by this type of interface. The former interface is retained and referred to as the 24-bit interface. Data areas and instructions use 24-bit addresses for the 24-bit interface and 31-bit addresses for the 31-bit interface. The 31-bit interface may be used in 24-bit addressing mode also. In the latter case, the addresses contained in the data area are interpreted as 24-bit addresses (i.e. the leftmost byte is not evaluated).

Macro description

The **GPARMOD** macro serves to assign, at program assembly time, one of the values 24 or 31 to the global Assembler variable &SYSMOD. &SYSMOD is evaluated at the time those macros are generated which allow for both a 24-bit and a 31-bit interface, but which were issued without the operand PARMOD=... being specified. Depending on the value of &SYSMOD either the 24-bit or the 31-bit interface is generated for the subsequent macros. Note that any specification of the PARMOD operand in a particular macro call prevails over the value in &SYSMOD at macro generation time.

Macro format and description of operands

GPARMOD
[24 / 31]

24
The 24-bit interface is generated for the subsequent macros. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated for the subsequent macros. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Examples

MAC1, MAC2 and MAC3 refer to macros which can generate both a 24-bit and a 31-bit interface.

```
    :
    :
GPARMOD 31 ———————————————————————————————————————————————————————— (1)
MAC1    op1, ..., op4
MAC2    op1, op2, op3
MAC3    op1, ..., op7
    :
    :
```

```
    :
    :
GPARMOD 24 ———————————————————————————————————————————————————————— (2)
MAC1    op1, ..., op4
MAC2    op1, ..., PARMOD=31
MAC3    op1, ..., op7
    :
    :
```

```
    :
    :
GPARMOD 31 ———————————————————————————————————————————————————————— (3)
MAC1    op1, ..., PARMOD=24
MAC2    op1, ..., PARMOD=24
MAC3    op1, ..., op7
    :
    :
```

(1) &SYSMOD is set to 31. Since the subsequent macros are issued without the PARMOD operand specified, the 31-bit interface is generated for each of these macros.

(2) &SYSMOD is set to 24. For MAC1 and MAC3, the 24-bit interface is generated, for MAC2 the 31-bit interface.

(3) &SYSMOD is set to 31. For MAC1 and MAC2, the 24-bit interface is generated in accordance with the value specified for the PARMOD operand; for MAC3, the 31-bit interface is generated as specified for &SYSMOD.

GTIME – Get date and time

General

Application areas:	Requesting and accessing lists and tables; see page 155
Macro type:	Type S, MF format **3**: C/D/E/L/M form; see page 29

Macro description

The **GTIME** macro gives the following information:

- the current date and time of day, either the UTC ('Universal Time Coordinate', corresponding to Greenwich Mean Time) or LT (standard 'Local Time' of the country)
- monotony behavior for callers on different XCS network node computers can be requested at the same time
- the current weekday
- the time zone of the system (corresponds to the shift in time with respect to UTC) in hours and minutes
- the amount of the time shift for daylight saving time (summer time) in hours and minutes
- the current time shift with respect to the standard time (winter time) based on daylight saving time (summer time).
- the points at which daylight saving time (summer time) changes to standard time (winter time) and vice versa (administered with the **CTIME** macro)
- whether the system time is synchronized with an external reference and, if so, with which reference time source (e.g. radio clock)
- whether a time change is due in the next hour
- the current epoch for the TODR (see the “Introduction to System Administration” manual [10])

GTIME supplies this information in the data area in printable, binary or TODR/TODX form (for the output of local time (LT) or the Universal Time Coordinate (UTC) only).

Notes

- Because the functions of the **GTIME** macro are initiated via a subprogram interface rather than a SVC, a program calling **GTIME** must provide a save area 18 words long. The address of this save area must be loaded into register R13 before the macro call.

- Apart from checking the standard header, there is no validation of the operand list.

Macro format and description of operands

GTIME
MODE=LT / UTC ,FORMAT=ISO4 / BIN / TODR / TODX ,RESOLVE=SEC / MICROSEC ,LINKADR=*NONE / linkaddr ,DATE=NO / YES ,DAY=NO / YES ,TOD=NO / YES ,ZONE=NO / YES ,EXTREF=NO / YES ,CHDATE=NONE / NEXT / PREV ,CHD_ANNOUNCMNT=*NO / *YES ,XCS_MODE=*NO / *YES ,MF=D / E / L / C / M [,PARAM=addr / (r)] ,PREFIX=N / p ,MACID=TIG / macid

The operands are described in alphabetical order below.

CHD_ANNOUNCMNT=
Announces a conversion time by means of a flag.

***NO**
No (other) flag should be set.

***YES**
A flag for a time change is set in the time interval of one hour before the conversion.

The data area for the output of the flag has the following structure (macro expansion with MF=D and standard value for PREFIX):

```
NTIGGINF            DS    AL1     general_info
NTIGICNH            EQU   X'80'   chdate is expected in next
*                                 hour
NTIGRESERVED_7BITS  EQU   X'7F'   not yet used
NTIGFRES            DS    XL2     reserved
```

CHDATE=

Specifies the direction in which conversion times (CHDATEs) are to be sought.
For further information on conversion times see notes on page 534.

<u>NONE</u>
Default setting. No (other) conversion time should be sought.

NEXT
The next conversion time should be sought.
The search begins at the contents of the field `<prefix><macid>CHD`.

PREV
The previous conversion time should be sought.
The search begins at the contents of the field `<prefix><macid>CHD`.

DATE=

Determines whether the current date is output.

<u>NO</u>
No information is given about the current date.

YES
The current date (calendar day and Julian date) is transferred to a data area. The value of the FORMAT operand governs the area and format in which the information is transferred:

FORMAT=ISO4

The current date is transferred in the form `yyyy-mm-ddjjj` (both hyphens are included in the output). Where:

`yyyy`	Year (four digits)
`mm`	Month (two digits, with leading zero as necessary)
`dd`	Day (two digits, with leading zero as necessary)
`jjj`	Julian date: current day of the year (three digits, with leading zeros as necessary)

The area in the operand list for accepting the date has the following format (macro expansion with MF=D and default value for PREFIX):

```
* DATE IN ISO4 FORMAT (EXAMPLE :2012-01-20020)
NTIGDTI       DS    0XL16       date_iso4
NTIGDATE_UN   DS    0XL10       date union
*
NTIGDATE_1    DS    0XL10       date struct
NTIGDTIY      DS    CL4         year
NTIGDTI1      DS    CL1         hyphen1
NTIGDTIM      DS    CL2         month
NTIGDTI2      DS    CL1         hyphen2
NTIGDTID      DS    CL2         day
*
```

```
              ORG   NTIGDATE_UN
NTIGDTIC      DS    CL10        date_char
              ORG   NTIGDATE_UN+10
NTIGDTIJ      DS    CL3         Julian date
NTIGDTIB      DS    CL1         blank
NTIGDYID      DS    CL2         weekday in ISO4
```

FORMAT=BIN

Year, month, day and Julian date are transferred as integers in each case (in binary form). The area in the operand list for accepting the date has the following format (macro expansion with MF=D and default value for PREFIX):

```
* DATE IN BINARY FORMAT
NTIGDTB       DS    0XL16       date_bin
*
NTIGDATE_2    DS    0XL6        date
NTIGDTBY      DS    H           year
NTIGDTBM      DS    H           month
NTIGDTBD      DS    H           day
*
NTIGDTBJ      DS    H           Julian date
NTIGFILL_6    DS    XL6         fill for weekday
NTIGDYBD      DS    H           weekday bin.: MO=0, DI=1, ...
*                                             SO=6
```

DAY=

Determines whether the current weekday is output.

<u>NO</u>

No information is given about the current weekday.

YES

The current weekday is transferred to a field of the operand list. The value of the FORMAT operand governs the area and format in which the information is transferred:

FORMAT=ISO4

The first two letters of the current weekday (MO, DI, MI, DO, FR, SA, SO) are transferred to the following field of the operand list (macro expansion with MF=D and default value for PREFIX):

```
* DAY OF WEEK IN ISO4 FORMAT (EXAMPLE : MO)
NTIGDYI       DS    0XL2
```

FORMAT=BIN

The current weekday is transferred as an integer (in binary form) (0 for Monday, 1 for Tuesday,..., 6 for Sunday) in the following halfword of the operand list (macro expansion with MF=D and default value for PREFIX):

```
* DAY OF WEEK IN BINARY FORMAT
NTIGDYB       DS    0XL2
```

EXTREF=
Determines whether the information on the external reference of the system time should be supplied where it is available.

<u>NO</u>
No information will be suppliued.

YES
The information on any available external reference of the system time should be supplied.

The data area for the output of the information has the following structure (default values for PREFIX and MACID):

```
* FLAGS OF GTIME: TIME REFERENCE
NTIGFLG             DS     0XL4    flags of GTIME
NTIGTREF            DS     FL1     time_reference
*   _time_reference_s
NTIGNONE            EQU    0       no external time reference
NTIGSVPF            EQU    1       SVP radio clock reference
NTIGCHNF            EQU    2       Channel rad. cl. reference
NTIGDCET            EQU    3       DCE reference
NTIGXCST            EQU    4       XCS reference
NTIGSKPX            EQU    5       SKP-X reference
```

FORMAT=
Defines the form in which **GTIME** supplies the requested information.

<u>ISO4</u>
The information is provided in printable form.

BIN
The information is provided in binary form.

TODR / TODX
The information is taken from the contents of the TOD register and the setting for the epoch in the TOD register.
The operand values for DATE, DAY, TOD and ZONE must be “NO”. The MODE operand is evaluated, i.e. MODE=LT must be set if the local time is to be output in TODR or TODX format; otherwise the UTC is output in TODR format.
The differences between TODR and TODX are described in the section “System time administration” in the “Introduction to System Administration” manual [10] and also in the information on the CTIME macro on page 356.

LINKADR=
Specifies the manner in which the address of the entry point I@GTIME for the **GTIME** routine in the GET-TIME subsystem is supplied to the user program. If MF=E, LINKADR must be specified; in all other cases, specification of LINKADR has no effect.

***NONE**
Default value: during assembly the assembler generates an external reference for the I@GTIME entry point and this is resolved during linking via the autolink function of the BLS.

This value can be used when the module containing the **GTIME** call:

- is always linked and loaded with the dynamic binder loader DBL (in this case, **GTIME** in the E form is allowed to issue a V constant, which is supplied by the BLS during the load procedure)
  or
- is linked with the BINDER of the new BLS (see the “BINDER” manual [5]) under the BINDER statement SET-EXTERN-RESOLUTION RESOLUTION=STD.

linkaddr
Symbolic address (name) of a word in which the user has provided the address of the I@GTIME entry before the **GTIME** call.

The following example shows how the address of the of the I@GTIME entry is first supplied to the program by an appropriate **BIND** call in the R1 register and can then be transferred for the **GTIME** call into the word designated by “linkaddr”:

```
         LINK  MF=E,PARAM=BINDPL
         :
         GTIME MF=E,PARAM=OPLIST,LINKADR=AENTRY
         :
AENTRY   DS    F
OPLIST   GTIME MF=L,...
BINDPL   BIND  MF=L,SYMBOL=I@GTIME,SYMBLAD=AENTRY
```

The entry address of the **GTIME** routine must always be supplied to the user program in this way if none of the cases mentioned in LINKADR=*NONE apply, e.g. especially if the module with the **GTIME** call is linked by BINDER under the BINDER statement SET-EXTERN-RESOLUTION RESOLUTION=MANDATORY.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see page 29).

In the E form of the macro, the label of the data area is specified in the PARAM operand.
Default setting: NTIG$PL
When calling the macro with MF=L, the user must specify this label explicitly, otherwise an `MNOTE` is output.

MODE=
Defines in which time system the date and time are to be output.

<u>LT</u>
Date and time are output in local time system (legal zone time).

UTC
Date and time are given as the UTC (Universal Time Coordinate, corresponds to Greenwich Mean Time).

RESOLVE=
specifies the desired level of expansion for the **GTIME** time specification for the formats ISO4 and BIN. For the format of output areas, see the areas specified for the TOD=YES operand.

<u>SEC</u>
Time is specified down to seconds.

MICROSEC
Time is specified down to microseconds.

TOD=
Determines whether the current time is output.

<u>NO</u>
No information is transferred about the current time.

YES
The current time is transferred to an area of the operand list. The value of the FORMAT operand governs the area and format in which the information is transferred:

FORMAT=ISO4

The current time is transferred in the form `hh:mm:ss` (both colons are included in the output). Where:

`hh`	Hours (two digits with leading zero as necessary)
`mm`	Minutes (two digits with leading zero as necessary)
`ss`	Seconds (two digits with leading zero as necessary)

The area in the operand list for accepting the time has the following format (macro expansion with MF=D and default value for PREFIX):

```
* TIME OF DAY WITH FORMAT=ISO4 (EXAMPLE : 08:31:09)
NTIGTDI      DS    0XL8        _tod_iso4_mdl
NTIGTDIH     DS    CL2         hour
NTIGTDI1     DS    CL1         colon1
NTIGTDIM     DS    CL2         minute
NTIGTDI2     DS    CL1         colon2
NTIGTDIS     DS    CL2         second
```

```
* SECOND FRACTION OF TIME OF DAY IN ISO4 FORMAT
* (EXAMPLE : .123456)
NTIGTFI      DS    0XL6        _ftod_iso4_mdl
NTIGTFIM     DS    CL3         millisecond
NTIGTFIN     DS    CL3         microsecond
```

FORMAT=BIN

Hour, minute and second are transferred as an integer in each case (in binary form). The area in the operand list for accepting the time has the following structure (macro expansion with MF=D and default value for PREFIX):

```
* TOD IN BINARY FORMAT
NTIGTDB      DS    0XL6        _tod_bin_mdl
NTIGTDBH     DS    H           hour
NTIGTDBM     DS    H           minute
NTIGTDBS     DS    H           second

* SECOND FRACTION OF TOD IN BINARY FORMAT
NTIGTFB      DS    0XL4        _ftod_bin_mdl
NTIGTFBM     DS    H           millisecond
NTIGTFBN     DS    H           microsecond
```

XCS_MODE=

supplies a time value which is strictly monotone within the XCS cluster with regard to a DLM lock for a shared resource.

***NO**

The monotony described above is not required.

***YES**

outputs a time value which ascends monotonically within an XCS cluster under the following conditions:
The caller must hold a DLM lock (see section “Distributed Lock Manager (DLM)” on page 140) at the time of the call. The monotony relationship applies only for the time stamp which has been fetched under the same lock.
The time value can be used for logging entries.
Its use is limited to the output format FORMAT=*TODR and MODE=*UTC. The necessary expansion is achieved in TODR format only. Restriction to MODE=*UTC ensures that monotony is maintained for all node computers even at the moment when a time change takes place.

ZONE=

Determines whether characteristic values of the local time zone are output.

NO

No information on the local time zone is transferred.

YES
Characteristic values of the local time zone are transferred to an area in the operand list. The value of the FORMAT operand governs the area and format in which the information is transferred:

FORMAT=ISO4
The characteristic values of the local time zone are transferred in the form `shh1:mm1-hh2:mm2-z` (colons and hyphens are included in the output). Where:

`s` “+” or “-”: Sign of time difference between the local time zone and UTC (Universal Time Coordinate, corresponds to Greenwich Mean Time).

`hh1:mm1` Time difference between the local time zone and UTC in hours (hh) and minutes (mm) (hh and mm are two digits in each case, with leading zeros as necessary).

`hh2:mm2` Time shift in the local time zone between daylight saving time (summer time) and standard time (winter time) in hours (hh) and minutes (mm) (hh and mm are two digits in each case, with leading zeros as necessary).

`z` “W” or “S”: Current timing in the local time zone (W for standard time (winter time), S for daylight saving time (summer time)).

The area in the operand list for accepting this information has the following format (macro resolution with MF=D and default value for PREFIX):

```
* ZONE IN ISO4 FORMAT (EXAMPLE :+08:00-01:00-S)
NTIGZOI      DS    0XL14       _zone_iso4_mdl
*
NTIGZOIC     DS    0XL6        time_zone
NTIGZOIS     DS    CL1         sign
NTIGZOIH     DS    CL2         hour
NTIGZOI1     DS    CL1         colon1
NTIGZOIM     DS    CL2         minute
*
NTIGZOI2     DS    CL1         hyphen1
*
NTIGZSIC     DS    0XL5        seasonal_difference
NTIGZSIH     DS    CL2         hours
NTIGZSI1     DS    CL1         colon1
NTIGZSIM     DS    CL2         minutes
*
NTIGZSI2     DS    CL1         hyphen2
NTIGZSIA     DS    FL1         actual season
*   _season_iso4_s
NTIGZSIW     EQU   230         'W': Wintertime
NTIGZSIS     EQU   226         'S': Daylight Savings Time
```

FORMAT=BIN

The characteristic values of the local time zone are transferred as integers in each case (in binary form).
The area in the operand list for accepting this information has the following format (macro resolution with MF=D and default value for PREFIX):

```
* ZONE IN BINARY FORMAT
NTIGZOB                DS    0XL10   _zone_bin_mdl
*
NTIGTIMEZONE           DS    0XL4    time_zone
NTIGZOBH               DS    H       hour
NTIGZOBM               DS    H       minute
*
*
NTIGSEASONAL_DIFFERENCE DS   0XL4    seasonal_difference
NTIGZSBH               DS    H       hours
NTIGZSBM               DS    H       minutes
*
NTIGZSBA               DS    FL1     actual season
*   _season_bin_s
NTIGZSBW               EQU   0       Wintertime
NTIGZSBS               EQU   1       Daylight Savings Time
*
NTIGZONE_BIN_FILL      DS    XL1     to fill the gap
```

Notes on conversion times

In order to use **CTIME** functionality with non-system conversion times, it must be possible to find out the conversion times of the system on which time data is created. BS2000 can administer up to 400 conversion times. One conversion time inquiry can be made with each **GTIME** macro call.
Information about conversion times is provided in a field of 8 bytes (`<prefix><macid>CHD`). The values specified for the CHDATE operand determine the direction of the conversion time inquiry. The reference point is the information present in this field when **GTIME** is called:
If all 8 bytes of the field contain X'00', the search begins after the next or preceding conversion time for the current system time. If the field already contains a date, however, this is interpreted as CHDATE information already supplied by **GTIME** and the search begins after that date.
The CHDATE information appears as STCK values based on UTC shifted 8 Bits logically to the right (SRDL instruction). The lowest-value bit indicates the type of conversion: 0 indicates conversion from standard time (winter time) to daylight saving time (summer time), and 1 daylight saving time (summer time) to standard time (winter time).

The **CTIME** function requires this data to be arranged as a table. The format of the table is given in the **CTIME** macro description, operands CHDLxIN and CHDLOUT. An example of working with conversion times is also given for the **CTIME** macro (see page 356).

Data returned to the data area can be accessed with the following names (default values for PREFIX and MACID):

```
NTIGCHD             DS    0XL8    change date
*
NTIGTODC            DS    0XL8    s_todr
NTIGCHDATE_UN       DS    0XL8    chdate_un
NTIGCHDATE_VALUE    DS    XL8     chdate_bit64
                    ORG   NTIGCHDATE_UN
*
NTIGCHDATE_MDL      DS    0XL8    chdate_mdl
NTIGCHD1            DS    F       most significant word
NTIGCHD2            DS    F       least significant word
```

Register contents

The following registers are required for a **GTIME** macro call:

R1 is loaded by the macro with the address of the operand list.

R13 before the macro call, is to be loaded with the address of an 18-word save area which the calling program has to provide.

R14 is loaded by the macro with the return address of the user program.

R15 is overwritten by the routine called (through **GTIME**).

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the GTIME macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

cc	bb	aaaa	Meaning
00	00	0000	Function executed successfully.
02		0010	The function was executed but the system knows of no CHDATE earlier than that specified in the data area. There is no point in calling CHDATE=PREV again as it will yield the same result. The 8 bytes of CHDATE information contain X'00..0001'.
02		0011	The function was executed but the system knows of no CHDATE later than that specified in the data area. There is no point in calling CHDATE=NEXT again as it will yield same result. The 8 bytes of CHDATE information contain X'00FFFFFFFFFFFFss', where 'ss' stands for the SEASON after the previous CHDATE (i.e. is X'00' or X'01').
02		0012	The function was executed but the system knows of no CHDATE. There is therefore no time change in the system. There is no point in calling the GTIME macro again.
	20	0008	The return code is possible only if XCS_MODE=*YES is specified. The function could not be executed. An internal error occurred when monotone time was determined for all XCS systems. cc refers to the error type but is only relevant for diagnosis within the manufacturer. There is no point in calling the GTIME macro again with XCS_MODE=*YES.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

The calling program is terminated when the following errors occur:

– The data area is not assigned to the caller.
– The data area is not aligned on a word boundary.
– The data area is protected against write access.

The TODR’s current epoch is supplied with every GTIME call in the <prefix><macid>EPD data field (one byte), see the section “System time administration” in the “Introduction to System Administration” manual [10].

For **examples** see section “S-type macros” on page 38 and the **CTIME** macro (page 356).

ILEMGT – Management of Indirect Linkage Entries (ILEs)

General

Application area:	Linking and loading; see page 47
Macro type:	Type S, MF format **2**: standard/C/D/E/L/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **ILEMGT** macro enables the user to manage a list of ILEs (Indirect Linkage Entries). The ILEs in the list may be generated, updated or deleted.

Information on ILEs can be requested with the **VSVI1** macro.

Macro format and description of operands

ILEMGT
MF=S / D / C / E / L / M ,ACTION=*CREATE / *UPDATE / *DELETE ,CONTEXT_NAME= '␣' / <c-string 1..32> / <var: char 1..32> ,CONTEXT_STATE=*DBL-OPTIONS / *ANY / *NEW / *OLD ,ILE_LIST_ADDR=NULL-1 / <var: pointer> ,ILE_LIST_LEN=0 / <integer 0..2147483647> ,MPID_ADDR=NULL-1 / <var: pointer> ,PARAM=<var: pointer> / (reg: pointer>) ,PREFIX=P / p ,MACID=ILE / macid

The operands are described in alphabetical order below.

ACTION=
Action to be executed for the ILEs in the list.

***CREATE**
The ILEs are to be created.

***UPDATE**
The ILEs are to be updated.

***DELETE**
The ILEs are to be deleted.

CONTEXT_NAME=

'␣' / <c-string 1..32> / <var: char 1..32>
Name of the context to which the ILEs belong.

CONTEXT_STATE=
State of the context specified in CONTEXT_NAME

***DBL-OPTIONS**
The parameter value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, the value that follows *DBL-OPTIONS in the syntax definition applies.

***ANY**
If the context already exists it is used, otherwise a new context is created.

***NEW**
The context is being created. It must not already exist.

***OLD**
The context must already exist.

ILE_LIST_ADDR=NULL-1 / <var: pointer>
Address of a list of ILEs that is to be transferred to DBL.

ILE_LIST_LEN=0 / <integer 0..2147483647>
Length of the ILE list (in bytes)

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM, PREFIX and MACID, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
It is possible to specify a PREFIX in the C form, D form, or M form of the macro, and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

MPID_ADDR=NULL-1 / <var: pointer>
Address of a field with the identifier of the memory pool containing the specified context. This identifier is made available to the user by means of the **ENAMP** macro.

ILE list format

The addresses specified in ILE_LIST_ADDR point to an ILE list which may contain a number of entries. An individual ILE entry has the following format:

Byte	Length	Field name	Meaning and/or value
0	8	HDR	Standard header
8	1	STATE	ACTIVE (X'01') or NOT_ACTIVE (X'02')
9	1	CONTROL	SYSTEM (X'01') or USER (X'02')
10	1	HSI_CODE	390 (X'01') or x86E (X'09')
11	1	RESERVED1	reserved (must contain X'00')
12	4	LOAD_ADDR	Address of the IL routine
16	4	SERVER_ADDR	Address of the ILE server
20	2	REF_DISPL	Distance of the external reference to the server within the IL routine
22	32	NAME	Name of the ILE symbol
54	2	RESERVED2	reserved (must contain X'0000')

The DSECT for such an ILE list entry is generated with `ILEMIT MF=D`.

Notes on the macro call

– If an address was specified with MPID_ADDR, the first character of the context name (mandatory parameter) determines whether or not the context is shared. If the Context name begins with „#“, DBL searches for the context in common memory pools and generates the context if necessary (**ASHARE** macro). In this case the MPID must identify a memory pool which is either already used with **ASHARE**, or which is blank (i.e. in which no memory is yet occupied).

  If the context name does not begin with '#', DBL searches in the normal user contexts. A context whose name begins with a letter may not be shared. DBL recognizes this as a conflict between the memory pool usage type and the name convention, and the macro is rejected with the corresponding return code.

– The number of memory pools in which the user can store shared code is limited to 16 per scope for each user ID, see page 219.

– If ACTION = *CREATE / *UPDATE is specified the STATE field of an ILE entry contains ACTIVE, and both LOAD_ADDR and SERVER_ADDR must refer to allocated memory areas.

– If the CONTROL field contains SYSTEM, DBL executes the following actions:

  a) When the ILE server is loaded
  the STATE field is set to ACTIVE and the address of the ILE server is entered into the IL routine.

  b) When the ILE server is unloaded
  the STATE field is set to NOT_ACTIVE and X'FFFFFFFF' is entered into the IL routine as the address of the ILE server.

  The CONTROL=SYSTEM field setting may not be used in combination with the ACTION=*UPDATE macro parameter.

– If ACTION=*CREATE is specified the LOAD_ADDR field must refer to an allocated memory area containing the user-defined IL routine (CONTROL=USER). If this does not apply (CONTROL=SYSTEM) then the DBL generates a standard IL routine in the context. The standard IL routine is generated in the HSI code that was specified when the **ILEMIT** macro was called, or else in the HSI code of the server on which the program is running.
If a user-defined IL routine is present, the REF_DISPL field must contain the distance of the word in the IL routine that refers to the ILE server. This word must be writable. Specification of the HSI code is mandatory for user-defined routines.

– If ACTION=*DELETE is specified, the IL routine will also be deleted from the context provided the ILE to be deleted was generated with CONTROL=*BY-SYSTEM.
The IL routine must have been generated by the DBL.

– All non-masked CSECTs and ENTRYs in the target context may be used as ILE servers, even those introduced by means of the **ETABLE** macro. To avoid any problems which this might cause for ILE management, one context should be set up for ILEs and their servers, and a different context set up for ETABLE symbols.

– If LLMs formed from PUBLIC and PRIVATE slices are to be used for indirect linking, these LLMs must be available in format 1 (see the “BINDER” manual [5]).

– The IL routines should have the attributes AMODE and RMODE=ANY and be loaded below 16 Mb, to ensure that they can be accessed by all programs.

– When an ILE is updated a check is performed when its server is loaded to see whether the HSI code and AMODE of ILE and server match. If not, loading of the server is rejected.

Return information and error flags

If an error occurs during processing of the list entries, a corresponding return code is entered in the HDR field of the list entry containing the error. The return code FUNCTION_PARTIALLY_PROCESSED is transferred in the standard header of the parameter list and processing of the list entries is continued..

The PROCESSED_ITEMS field of the parameter list contains the number of correctly processed entries.

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the ILEMGT macro is transferred in the standard header
(cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally.
X'02'	X'00'	X'0001'	The function was partially executed.
X'61'	X'01'	X'0001'	Invalid value in ILE_LIST_LEN.
X'61'	X'01'	X'0002'	Invalid parameter combination in the parameter list.
X'61'	X'01'	X'0003'	The MESSAGE parameter is invalid.
X'61'	X'01'	X'0004'	The context name is invalid.
X'61'	X'01'	X'0008'	The ACTION parameter is invalid.
X'61'	X'40'	X'0011'	The task is not connected to the memory pool.
X'61'	X'01'	X'0014'	The new USER context already exists in another memory pool.
X'61'	X'40'	X'0015'	Maximum number of contexts in the memory pool has already been reached.
X'61'	X'40'	X'0016'	The maximum number of memory pools with this scope has already been reached for the user ID. No more memory pools can be used with this scope.
X'61'	X'40'	X'0017'	System storage area for user contexts is full.
X'61'	X'01'	X'0018'	A reserved field does not contain any binary zeros.
X'61'	X'01'	X'0019'	The current memory pool is not shared.
X'61'	X'20'	X'0100'	System error (e.g.: $REQM, $RELM, $CSTA, RDTFT)
X'61'	X'20'	X'0101'	Internal DBL error.
X'61'	x'80'	X'0103'	Shared resources are not available.
X'61'	X'01'	X'0114'	CONTEXT_STATE=*OLD was specified and the context does not exist.
X'61'	X'01'	X'0118'	CONTEXT_STATE=*NEW was specified and the specified context already exists.
X'61'	X'40'	X'0120'	MPID_ADDR is not aligned on a word boundary.

X'cc'	X'bb'	X'aaaa'	Meaning
X'61'	X'01'	X'0134'	The CONTEXT_STATE parameter is invalid.
X'61'	X'40'	X'0148'	An attempt was made to use a context which has been corrupted by a previous error.
X'61'	X'40'	X'0158'	Maximum number of 16 user contexts has been reached. No more new contexts can be generated.
X'61'	X'01'	X'0184'	The specified memory pool is invalid.
X'61'	X'20'	X'0198'	Insufficient memory availabe for the action requested.
X'61'	X'20'	X'0204'	Inconsistencies in the DBL memory management tables (system error).
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported.
X'00'	X'03'	X'FFFF'	The interface version is not supported.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

ILEMIT – Generate or update a list entry for an ILE list

General

Application area:	Linking and loading; see page 47
Macro type:	Type S, MF format **2**: standard/C/D/L/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **ILEMIT** macro generates or updates a list entry for an ILE list which is used in the **ILEMGT** macro.

Macro format and description of operands

ILEMIT
MF=D / C / L / M ,CONTROL=*NOT-SPECIFIED / *BY-SYSTEM / *BY-USER ,HSI_CODE=*BY-SYSTEM / *390 / *RISC [1] ,ILE_NAME=<c-string 1..32> / <var: char 1..32> ,LOAD_ADDR=NULL-1 / <var: pointer> ,REF_DISPL=0 / <integer 0..65535> ,SERVER_ADDR=NULL-1 / <var: pointer> ,STATE=*NOT-SPECIFIED / *ACTIVE / *NOT-ACTIVE ,PARAM=<var: pointer> / (reg: pointer>) ,PREFIX=P / p] ,MACID=ILT / macid

[1] The operand value *RISC has no meaning in BS2000/OSD-BC V9.0 and higher

The operands are described in alphabetical order below.

CONTROL=
Specifies whether ILE is to be controlled by DBL or by the user, i.e. the person editing the IL routine.

*NOT-SPECIFIED
This value is not permitted during generation of a list entry.

***BY-SYSTEM**
The DBL generates the standard IL routine.

***BY-USER**
There is a user-defined IL routine.

HSI_CODE=
Indicates in which of the servers machine codes the IL routine exists. This operand is mandatory if LOAD_ADDR is specified.

***BY-SYSTEM / *390**
The default setting is the server type on which the IL routine is running.

ILE_NAME=<c-string 1..32> / <var: char 1..32>
Name of the ILE symbol.

LOAD_ADDR=NULL-1 / <var: pointer>
Address of the IL routine.
If this operand is specified the HSI_CODE operand must also be specified.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM, PREFIX and MACID, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
It is possible to specify a PREFIX in the C form, D form, or M form of the macro, and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

REF_DISPL=0 / <integer 0..65535>
Distance of the server address word within the IL routine.

SERVER_ADDR=NULL-1 / <var: pointer>
Address of the server (only if STATE=ACTIVE is specified).

STATE=

***NOT-SPECIFIED**
Indicates whether the ILE server is loaded. This value is not permitted during the generation of a list entry.

***ACTIVE**
The server is loaded.

***NOT-ACTIVE**
The server is not loaded.

Notes on the macro call

The notes relating to the **ILEMGT** macro also apply to the **ILEMIT** macro. The layout of a list entry is also described there.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the ILEMIT macro is transferred in the standard header
(cc=subcode2, bb=subcode1, aaaa=main code):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally.
X'61'	X'01'	X'000A'	The ILE entry is invalid.
X'61'	X'01'	X'0012'	ILE not found (if ACTION=*DELETE or ACTION=*UPDATE is specified).
X'61'	X'01'	X'0013'	ILE already exists (if ACTION=*CREATE is specified).
X'61'	X'01'	X'0018'	A reserved field does not contain any binary zeros.
X'61'	X'01'	X'0020'	An invalid value has been set for STATE, CONTROL or HSI_CODE.
X'61'	X'01'	X'0021'	Requested action cannot be executed with this CONTROL setting.
X'61'	X'01'	X'0022'	Server address word (REF_DISPL) in the IL is not writable.
X'61'	X'01'	X'0023'	The user-specific IL routine is not loaded.
X'61'	X'01'	X'0024'	Operands (STATE or CONTROL) missing from list entry generation.
X'61'	X'01'	X'002C'	The ILE name is invalid. It must begin with a blank.
X'61'	X'40'	X'0050'	HSI codes of ILE and ILE server do not match: ILE was not generated.
X'61'	X'01'	X'0130'	Invalid address (LOAD_ADDR, SERVER_ADDR) in current list entry.
X'61'	X'40'	X'0194'	Error in validation of a class 6 memory area.
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported.
X'00'	X'03'	X'FFFF'	The interface version is not supported.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

IOSID – Request operating system identification and version

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Type O; see page 28

Macro description

The **IOSID** macro supplies the ID and version number of the operating system. The information is entered in the global text variable &IOSID or in register R1. The entry has one of the following formats:

`C'x'`	in &IOSID
`C'xvvv'`	in register R1

where:

```
x=2:  BS2000
vvv:  version number, e.g. 190 ≙ V19.0
```

The entry in &IOSID is made during macro assembly and depends on the macro library in use.

For information on global text variables (variable parameters), see the “ASSEMBH” manual [2].

Macro format and description of operands

IOSID
[GBLC]

GBLC
Transfers the operating system ID to the global text variable &IOSID.

without operand
The operating system ID and the version number are transferred to register R1.

Example

```
IOSID    START
         PRINT NOGEN
*
*    Definition of macro QUERY
*
         MACRO ────────────────────────────────────────────────────────── (1)
&NAME    QUERY
         GBLC  &IOSID
&NAME    NOP   &NAME
         AIF   ('&IOSID'(1,1) EQ '2').BS2 ─────────────────────────────── (2)
         MNOTE 9,'Only BS2000 possible'
.BS2     WROUT MESS,ERROR
ERROR    B     END
MESS     DC    Y(ENDM-MESS)
         DC    X'404001'
         DC    C'Operating system BS2000'
ENDM     EQU   *
END      NOP   &NAME
         MEND
*
*        Program start
*
         BALR  3,0
         USING *,3
         IOSID ────────────────────────────────────────────────────────── (3)
         ST    1,WORD
         MVC   GR1CONT,WORD
         WROUT GR1OUT,END1
         IOSID GBLC ───────────────────────────────────────────────────── (4)
CALLMAC  QUERY ────────────────────────────────────────────────────────── (5)
END1     TERM
GR1OUT   DC    Y(GR1END-GR1OUT)
         DC    X'404001'
         DC    C'Reg1: '
GR1CONT  DS    CL4
GR1END   EQU   *
WORD     DS    F
         END
```

(1) The QUERY macro is generated to analyze the contents of &IOSID:
- Issue message (WROUT) if &IOSID contains the character “2” (BS2000).
- MNOTE message for all other cases.

(2) The global text variable &IOSID is requested.

(3) Operating system ID and version are to be output in register R1.

(4) The operating system ID is to be output to the global text variable &IOSID.

(5) Call the QUERY macro.

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,iosid), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,iosid))
%  ASS6011 ASSEMBLY TIME: 336 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 87 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=iosid
%  BLS0523 ELEMENT 'IOSID', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'IOSID', VERSION ' ' OF '<date> <time>' LOADED
Reg1: 2190 ────────────────────────────────────────────────────────────  (6)
Operating system BS2000 ───────────────────────────────────────────────  (7)
```

(6) Output of the edited register R1 after calling the macro **IOSID** without operands. The following information is contained:

Operating system: BS2000
Version: 19.0

(7) After calling the macro **IOSID** with the GBLC operand, the result of the evaluation from the QUERY will be output.

JINF – Request job information

General

Application areas: Requesting and accessing lists and tables; see page 155
Communication; see page 163
Macro type: Type S, MF format **1**: 24-bit interface: standard/E/L form
31-bit interface: standard form; see page 29

A job begins with the SET-LOGON-PARAMETERS command and ends with the EXIT-JOB command. A job can also be created by transferring a procedure file to the system using the ENTER-PROCEDURE command. A job is categorized in a job class by the operating system according to its job data (attributes). Both user-specific (accounting) data and job-specific data (job name, job class, job priority, start time,...) are included in the job data. The user transfers the latter specifications to the operating system in the SET-LOGON-PARAMETERS or ENTER-JOB or ENTER-PROCEDURE commands.

Macro description

The **JINF** macro transfers a list of job data to an area to be specified. The list is subdivided into data that is specified in the SET-LOGON-PARAMETERS or ENTER-JOB command and data on the current job (job start time, number of job repetitions). The **DJINF** macro generates a description (DSECT/data section) of the output list. The lengths of the areas to be created should be taken from this description.
In 31-bit addressing mode, the area to be created (PARLIST=...) must start with the standard header.

Macro format and description of operands

JINF
[PARMOD=24] [,JATTR=addr / (r)] [,RUNTIME=addr / (r)] [,MF=L / (E,..)] [PARMOD=31],PARLIST=addr / (r)

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses. (Address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses. (Address space ≤ 2 Gb). Data lists start with the standard header.

JATTR=
Defines the address of an area to receive the job data at job generation time. The area should be aligned on a word boundary.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

RUNTIME=
Defines the address of an area to receive the current job data (job start time, number of job repetitions). The area should be aligned on a word boundary.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

PARLIST=
Gives the address of an area to receive the job data. The area should be aligned on a word boundary and must start with the standard header. When using the data list created by means of **DJINF**, the standard header is initialized automatically; in all other cases, it is the user's task to initialize the standard header.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:

0	0	a	a	a	a	a	a

A return code relating to the execution of the JINF macro is transferred in register R15.

X'aaaaaa'	**Meaning**
X'000000'	Normal execution.
X'000008'	Operand error or insufficient length or PARLIST specified with PARMOD=24.
X'00000C'	System error.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Example

```
JINF    START
JINF    AMODE ANY
        PRINT NOGEN
        BALR  3,0
        USING *,3
        JINF  PARLIST=JIJOBDPL,PARMOD=31 ——————————————————————————————  (1)
        MVC   EMPF1,JIJOUID ———————————————————————————————————————————  (2)
        MVC   EMPF2,JIJOACC
        MVC   EMPF3,JIJOJCLA
        MVC   EMPF4,JIJOTSN
        WROUT AUS1,ERROR,MODE=LINE,PARMOD=31
TERM    TERM
ERROR   TERM  DUMP=Y
        DS    0F
AUS1    DC    Y(AUS1END-AUS1)
        DS    CL3
        DC    C'USER ID:  '
EMPF1   DS    CL8
F1      DC    X'15'
        DC    C'ACCOUNT NUMBER: '
EMPF2   DS    CL8
F2      DC    X'15'
        DC    C'JOB CLASS:  '
EMPF3   DS    CL8
F3      DC    X'15'
        DC    C'TSN:  '
EMPF4   DS    CL4
AUS1END EQU   *
        DJINF DSECT=NO,PARLIST=YES ————————————————————————————————————  (3)
        END
```

This program, in conjunction with the use of the symbolic names in the data list (macro **DJINF**) output certain job data (user ID, account number, job class, task sequence number = TSN). The program is executed in 31-bit addressing mode.

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,jinf), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,jinf))
%  ASS6011 ASSEMBLY TIME: 344 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 81 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=jinf
%  BLS0523 ELEMENT 'JINF', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'JINF', VERSION ' ' OF '<date> <time>' LOADED
USER ID:  QM212    ————————————————————————————————————————————————————  (4)
ACCOUNT NUMBER: 89002
JOB CLASS:  JCDSTD
TSN:  2QSE
```

(1) The **JINF** macro is called.

(2) Transfer the job data to the output field and output it line by line with the **WROUT** macro.

(3) The **DJINF** macro is called to generate the data list (output area). The initialization values for the standard header are entered.

(4) Line-by-line output to SYSOUT of the selected job data.

JMGDJP – Create DSECT or data area for JMGJPAR macro

General

Application areas:	Requesting and accessing lists and tables; see page 155
Macro type:	Definition macro; see page 28

Macro description

The **JMGDJP** macro generates a description (DSECT/data area) of the operand list of the **JMGJPAR** macro. DSECT and data area start with the standard header, the initialization values are entered.

Macro format and description of operands

JMGDJP
DSECT=YES / NO [,PREFIX=p]

DSECT=
Indicates whether a dummy section (DSECT) for the operand list is to be generated or whether data area and definitions are to be created in the user program directly.

- **YES**
  A dummy section is generated.

- **NO**
  Data area and definitions are to be created in the user program directly.

PREFIX=
Gives a character string to be prefixed to the symbolic names of the DSECT or the data area.

- **p**
  Prefix for the symbolic names. Length ≤ 2 characters.
  Default value: p=JP.

JMGJPAR – Request job parameters

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Type R; see page 28

Additional attributes (job parameters) relating to the job class can be specified in the SET-LOGON-PARAMETERS, ENTER-JOB and ENTER-PROCEDURE commands if they were previously defined and made known by system administration (see the ENTER-JOB, ENTER-PROCEDURE and SET-LOGON-PARAMETERS commands in the “Commands” manual [19]).

Macro description

The **JMGJPAR** macro transfers the job parameters specified in the SET-LOGON-PARAMETERS, ENTER-JOB or ENTER-PROCEDURE command to a specified area. The **JMGDJP** macro generates a description (DSECT/data list) of the output area.

Macro format and description of operands

JMGJPAR
PARLIST=addr / (r)

PARLIST=
Specifies the address of an area to receive the job parameters. The area should be aligned on a word boundary and must start with the standard header. The standard header is initialized if and when the area is created with the **JMGDJP** macro.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:

0	0	a	a	a	a	a	a

A return code relating to the execution of the JMGJPAR macro is transferred in register R15.

X'aaaaaa'	Meaning
X'000000'	Normal execution.
X'000004'	No job parameters specified.
X'000008'	Operand error.
X'00000C'	System error.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

JOBINFO – Request job information

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Type S, MF format **2**: standard/C/D/L/E form; see page 29

- The **JOBINFO** macro is an extension of the **JINF** macro in that it supplies information about individual jobs selected by the user by means of their TSNs. Note that information about spoolout jobs cannot be obtained in this way.

A job begins with the SET-LOGON-PARAMETERS command and ends with the EXIT-JOB command. A job can also be created by transferring a procedure file to the system with the ENTER-PROCEDURE command. A job is categorized in a job class by the operating system according to its job data (attributes). Both user-specific (accounting) data and job-specific data (job name, job class, job priority, start time,...) are included in the job data. The user submits the latter specifications to the operating system in the SET-LOGON-PARAMETERS or ENTER-JOB / ENTER-PROCEDURE command.

Macro description

The **JOBINFO** macro transfers a list of job data to an output area. The job must have been submitted to the Job Management System by entering either the SET-LOGON-PARAMETERS, ENTER-JOB or ENTER-PROCEDURE command, i.e. the inquiry can refer to an interactive job, a batch job or a job in the job pool that has not yet been started. The job is identified via its TSN. Nonprivileged users only receive information about jobs running under their own user IDs or about jobs started under their own user IDs. The FORM operand serves to control the amount of information output.

FORM=SHORT supplies the following items of information:
TSN of the job, user ID, account number, job class, job name, job type (interactive job, batch job, job in job pool), requested CPU time, start attributes (SOON, EARLIEST,...,DAILY, WEEKLY,..), (requested) start date, (requested) start time.

FORM=LONG additionally supplies the following items of information:
retry interval and number of job repetitions, job priority, RERUN and FLUSH data, (real) start date, (real) start time, spoolin data, spoolin time, name of monitoring job variable, name of the SYSCMD (Enter) file, length of job parameter, job parameter, BCAM name (if the job is running on a computer other than the home computer), TSN of the task issuing the macro call.

The I/O area is created immediately following the standard header. It begins with the (input) field for the job's TSN. Entries in this field must be made dynamically whenever the C/D form of a macro is used. The input field is followed by the output area whose length can be obtained from a list generated with MF=C/D. Initialization values are entered in the standard header.

Macro format and description of operands

JOBINFO
[TSN='tsn' / addr / (r)] ,FORM=LONG / SHORT ,CONST=YES / NO ,MF=S / E / L / C / D [,PARAM=addr / (r)] ,PREFIX=J / p ,MACID=OBI / macid

TSN=
TSN (job number) of the job about which information is requested. Nonprivileged users can only specify jobs running under their own user IDs or that have been started in their own user IDs. System administration is allowed to request information about any job.

'tsn'
Job number (TSN). Default value: tsn='␣␣␣␣' (4 blanks): this means that job data for the calling program is output.

addr
Symbolic address (name) of the 4-byte field containing the TSN.

(r)
Register containing the TSN (right-justified).

FORM=
Controls the amount of information output, cf. macro description.

LONG
Full-length information output.

SHORT
Abbreviated information output.

CONST=

Determines whether equates are to be entered in the output list in order to facilitate the interpretation of field contents. CONST can be specified in conjunction with MF=C/D only.

YES

Equates are entered.

NO

No equates are entered.

MF=

For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form or D form of the macro and additionally a MACID in the C form (see section “S-type macros” on page 29).

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the JOBINFO macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function has been executed.
X'00'	X'02'	X'0001'	Job has not been submitted to the Job Management System; no job data could be found (specified TSN not assigned to any job).
X'00'	X'03'	X'0001'	Job has not been submitted to the Job Management System; no job data could be found (specified TSN is a SPOOL job).
X'00'	X'01'	X'0003'	System error: internal interface error
X'00'	X'02'	X'0003'	System error: job data not accessible
X'00'	X'03'	X'0003'	System error: no memory space available.
X'00'	X'00'	X'0004'	Authorization error: e.g. specified TSN assigned to different user ID.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

The following errors cause the calling program to be aborted:

- The data area has not been assigned to the calling program.
- The data area is not aligned on a word boundary.
- The data area is write-protected.

JSATTCH – Attach job scheduler to Job Management System

General

Application area:	Job scheduler (system administration macro); see page 165
Macro type:	Type S, MF format **1**: 24-bit interface: standard/E/L form 31-bit interface: standard form; see page 29

- The **JSATTCH** macro can only be called under the TSOS (system administration) ID.
- When using the 31-bit interface, the **DJSIPL** macro serves to generate a description (DSECT/data list) of the operand list; when using the 24-bit interface, the same function is performed by the **DJSI** macro.
- JMS = Job Management System; JSS = Job Scheduling Supports. JSS is part of the Job Management System.

Macro description

The **JSATTCH** macro indicates to JMS that the job scheduler is ready to process JSS events.
JMS confirms by returning a date entry in the form of the minutes that have elapsed since 1.1.1980 (0.00 hours).

Note

It is essential to use this macro in order that JSS can support the job scheduler (receive job start requests, transfer JSS events).

Macro format and description of operands

JSATTCH
CLOCK=addr / (r) [,PARMOD=24] [,MF=L / (E,..)] [PARMOD=31] ,PARLIST=addr / (r)

CLOCK=
Defines the address of an area in which the date entry is made. The area must be aligned on a word boundary. This operand can be specified only for the 24-bit interface.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands, see page 29.
The valid MF values and default settings for this macro are given at the start of the macro description.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses. (Address space ≤ 16 Mb.)

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses. (Address space ≤ 2 Gb.) Data lists start with the standard header.

PARLIST=
Gives the address of the area containing the date entry. The area should be aligned on a word boundary and must start with the standard header. This operand can be specified only for the 31-bit interface.
The **DJSIPL** macro generates a description (DSECT/data list) of this area; the initialization values for the standard header are entered.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

Return information and error flags

R15:

0	0	a	a	a	a	a	a

A return code relating to the execution of the JSATTCH macro is transferred in register R15.

X'aaaaaa'	Meaning
X'000000'	Normal execution.
X'000008'	Operand error.
X'00000C'	System error.
X'000010'	Unauthorized caller (not TSOS).
X'000018'	Macro was carried out after a JSATTCH macro or job stream has already terminated.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

JSDETCH – Detach job scheduler from Job Management System

General

Application area:	Job scheduler (system administration macro); see page 165
Macro type:	Type S, MF format **1**: 24-bit interface: standard/E/L form 31-bit interface: standard form; see page 29

- The **JSDETCH** macro can only be called under the TSOS (system administration) ID.
- When using the 31-bit interface, the **DJSIPL** macro serves to generate a description (DSECT/data list) of the operand list.
- JMS = Job Management System; JSS = Job Scheduling Supports. JSS is part of the Job Management System.

Macro description

The **JSDETCH** macro indicates to the JMS that the job scheduler need no longer be supported by the system. The job scheduler is detached from JSS.
If **JSDETCH** is called without operands, the 24-bit interface is generated.

Macro format and description of operands

JSDETCH
{ PARMOD=24 [,MF=L / (E,..)] [PARMOD=31] ,PARLIST=addr / (r) }

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses. (Address space ≤ 16 Mb.)

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses. (Address space ≤ 2 Gb.) Data lists start with the standard header.

PARLIST=
Gives the address of the operand list. The list should be aligned on a word boundary and must start with the standard header. The **DJSIPL** macro generates a description (DSECT/data list) of the operand list; the initialization values for the standard header are entered.

addr
Symbolic address (name) of the operand list.

(r)
Register containing the address value “addr”.

Return information and error flags

When the 31-bit interface is used, register R1 is overwritten.

R15:	0	0	a	a	a	a	a	a

A return code relating to the execution of the JSDETCH macro is transferred in register R15.

X'aaaaaa'	Meaning
X'000000'	Normal execution.
X'000008'	Operand error.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

JSEXPCT – Request JSS events

General

Application area:	Job scheduler (system administration macro); see page 165
Macro type:	24-bit interface, type S: MF format **1** (standard/E/L form) 31-bit interface, type S: MF format **2** (standard form); see page 29

- The **JSEXPCT** macro can only be called under the TSOS (system administration) ID.
- When using the 31-bit interface, the **DJSIPL** macro serves to generate a description (DSECT/data list) of the operand list, of all JSS events, as well as EQUATES for the return code; when using the 24-bit interface, the same function is performed by the **DJSI** macro.
- JMS = Job Management System; JSS = Job Scheduling Supports.

  JSS is part of the Job Management System.

Macro description

The next event for the job scheduler is requested from JMS with the **JSEXPCT** macro. The events refer to a job (job acceptance, job termination, job RELEASE, job CANCEL, etc.) managed by the job scheduler, to the job class (job class in the HOLD state, resetting, reacceptance of jobs) or to the job scheduler (job scheduler in the HOLD state, resetting, modified STREAM-PARAMETER, ...).

Note

> The job scheduler is put into a wait state if there is no event for processing. When an event occurs, the job scheduler is removed from the wait state.

Macro format and description of operands

JSEXPCT
⎧ EVENT=addr / (r) [,PARMOD=24] [,MF=L / (E,..)] ⎫ ⎩ [PARMOD=31] ,PARLIST=addr / (r) ⎭

EVENT=
Defines the address of an area where the event is entered. The area must be aligned on a word boundary.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands, see page 29.
The valid MF values and default settings for this macro are given at the start of the macro description.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses. (Address space ≤ 16 Mb.)

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses. (Address space ≤ 2 Gb.) Data lists start with the standard header.

PARLIST=
Gives the address of the operand list. The list should be aligned on a word boundary and must start with the standard header. The **DJSIPL** macro generates a description (DSECT/data list) of the operand list; the initialization values for the standard header are entered.

addr
Symbolic address (name) of the operand list.

(r)
Register containing the address value “addr”.

Description of the events (24-bit interface/31-bit interface)

JSIEJINT/JSEXJINT: introduce a job
The job scheduler is requested to introduce the job to the set of jobs it manages. The following information on the job is supplied to the scheduler:

- TSN
- name of the job class
- CPU time
- job priority
- arrival time
- start type (SOON, IMMEDIATE, ...)
- start time
- hold state
- job parameters (J-PAR)

JSIEJTER/JSEXJTER: terminate execution of a job started by the job scheduler.

JSIEJHLD/JSEXJHLD: place a job that is waiting to be started in the hold state.
The job scheduler is requested to ignore the job during scheduling. A job in the hold state is marked as such in the job pool (EQUISAMQ file).

JSIEJREL/JSEXJREL: release a job from the hold state.
The job scheduler is requested to allow for the job again in scheduling.

JSIEJCAN/JSEXJCAN: delete a job that has not yet been started by the job scheduler.
The job scheduler is requested to cancel the job.

JSIEJEXP/JSEXJEXP: introduce a job that is to be started as soon as possible.

JSIEJRES/JSEXJRES: modify the attributes of a job that has not yet been started with the MODIFY-JOB command.

JSIECHLD/JSEXCHLD: place the specified job class in the hold state.
As in the case of JSIEJHLD/JSEXJHLD, the job scheduler is requested to ignore jobs of this class in future. All the jobs of a job class in the hold state are also marked as such in the job pool (EQUISAMQ file).

JSIECREL/JSEXCREL: release the job class from the hold state.
The job scheduler is requested to allow once again for jobs of this class during scheduling.

JSIECAVA/JSEXCAVA: CLASS-LIMIT has not been reached for the specified job class.
This occurs either when a job running in the class is terminated or by raising the CLASS-LIMIT with the MODIFY-JOB-CLASS command.

JSIESHLD/JSEXSHLD: the specified stream has been placed in the hold state.
The job scheduler is requested not to start any more jobs.

JSIESREL/JSEXSREL: release the hold state for the stream.
The job scheduler is requested to start jobs again.

JSIESCLQ/JSEXSCLQ: indicator that the operating system is in the shutdown state.
This event has the same effect as JSIESHLD/JSEXSHLD, i.e. no more jobs are started.

JSIESCLI/JSEXSCLI: causes the scheduler to terminate immediately (program termination). This event is initiated with the STOP-JOB-STREAM command. No more events are sent to the stream after this event.

JSIESCHA/JSEXSCHA: information about modified STREAM-PARAMETERs.

JSIETIM/JSEXTIM: event which occurs every minute and which enables the job scheduler to carry out time functions (e.g. support of the START operand in the SET-LOGON-PARAMETERS command and the REPEAT-JOB operand in the ENTER-JOB command).

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:

0 0	a a	a a	a a

A return code relating to the execution of the JSEXPCT macro is transferred in register R15.

X'aaaaaa'	Meaning
X'000000'	Normal execution.
X'000008'	Operand error.
X'00000C'	System error.
X'000010'	Unauthorized caller (not TSOS).
X'000018'	Macro called before a JSATTCH macro or after a JSDETCH.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

JSINFO – Access STREAM-PARAMETER values

General

Application area:	Job scheduler (system administration macro); see page 165
Macro type:	Type S, MF format **1**: 24-bit interface: standard/E/L form 31-bit interface: standard form; see page 29

- The **JSINFO** macro can only be called under the TSOS (system administration) ID.
- When using the 31-bit interface, the **DJSIPL** macro serves to generate a description (DSECT/data list) of the operand list, when using the 24-bit interface, the same function is performed by the **DJSI** macro.
- JMS = Job Management System; JSS = Job Scheduling Supports.

  JSS is part of the Job Management System.

Macro description

The STREAM-PARAMETER (S-PAR) values of the stream definition are accessed with the **JSINFO** macro.
The **JSINFO** macro should be called before the **JSATTCH** macro.

Macro format and description of operands

JSINFO

{ JSINF=addr / (r) [,PARMOD=24] [,MF=L / (E,..)] }
{ [PARMOD=31] ,PARLIST=addr / (r) }

JSINF=
Defines the address of an area in which the STREAM-PARAMETER values are entered. The area must be aligned on a word boundary.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses. (Address space ≤ 16 Mb.)

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses. (Address space ≤ 2 Gb.) Data lists start with the standard header.

PARLIST=
Gives the address of the operand list. The list should be aligned on a word boundary and must start with the standard header. The **DJSIPL** macro generates a description (DSECT/data list) of the operand list; the initialization values for the standard header are entered.

addr
Symbolic address (name) of the operand list.

(r)
Register containing the address value “addr”.

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:	0	0	a	a	a	a	a	a

A return code relating to the execution of the JSINFO macro is transferred in register R15.

X'aaaaaa'	Meaning
X'000000'	Normal execution.
X'000008'	Operand error.
X'000010'	Unauthorized caller (not TSOS).

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

JSRUNJB – Transfer job to start

General

Application area:	Job scheduler (system administration macro); see page 165
Macro type:	Type S, MF format **1**: 24-bit interface: standard/E/L form 31-bit interface: standard form; see page 29

- The **JSRUNJB** macro can only be called under the TSOS (system administration) ID.
- When using the 31-bit interface, the **DJSIPL** macro serves to generate a description (DSECT/data list) of the operand list; when using the 24-bit interface, the same function is performed by the **DJSI** macro.
- JMS = Job Management System; JSS = Job Scheduling Supports. JSS is part of the Job Management System.

Macro description

By means of the **JSRUNJB** macro the job scheduler requests the class scheduler to start the specified job.
The job is defined by

– its TSN,
– the name of the job class to which the job was assigned and
– start specifications for the job (start indicator).

Note

> If the CLASS-LIMIT has been reached for the job class, the job is not started. An appropriate return code is transferred.

Macro format and description of operands

JSRUNJB

{ PARMOD=24 ,STRTINF=addr / (r) [,MF=L / (E,..)] }
{ [PARMOD=31] ,PARLIST=addr / (r) }

STRTINF=
Defines the address of an area with the data of the job to be started. The area must be aligned on a word boundary.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses. (Address space ≤ 16 Mb.)

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses. (Address space ≤ 2 Gb.) Data lists start with the standard header.

PARLIST=
Gives the address of the operand list. The list should be aligned on a word boundary and must start with the standard header. The **DJSIPL** macro generates a description (DSECT/data list) of the operand list; the initialization values for the standard header are entered.

addr
Symbolic address (name) of the operand list.

(r)
Register containing the address value “addr”.

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:

0	0	a	a	a	a	a	a

A return code relating to the execution of the JSRUNJB macro is transferred in register R15.

X'aaaaaa'	Meaning
X'000000'	Normal execution.
X'000004'	TSN of job to be started is unknown.
X'000008'	Operand error.
X'00000C'	System error.
X'000010'	Unauthorized caller (not TSOS).
X'000018'	Macro called before a JSATTCH macro or after a JSDETCH.
X'00001C'	The job is not in Q1 or has not been marked as “accepted for scheduling” by the Job Management System.
X'000020'	Job class limit exceeded.
X'000024'	Job stream in HOLD state.
X'000028'	Job in HOLD state.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

JSWAKE – Initiate timer event for job scheduler

General

Application area:	Job scheduler (system administration macro); see page 165
Macro type:	Type S, MF format **1**: 24-bit interface: standard/E/L form 31-bit interface: standard form; see page 29

- The **JSWAKE** macro can only be called under the TSOS (system administration) ID.
- When using the 31-bit interface, the **DJSIPL** macro serves to generate a description (DSECT/data list) of the operand list; when using the 24-bit interface, the same function is performed by the **DJSI** macro.
- JMS = Job Management System; JSS = Job Scheduling Supports. JSS is part of the Job Management System.

Macro description

The **JSWAKE** macro serves to initiate the next timer event for the job scheduler. The time specification for the timer event is transferred in a field.

Macro format and description of operands

```
JSWAKE

 { PARMOD=24 ,JSWAKE= addr / (r) [,MF=L / (E,..)] }
 { [PARMOD=31] ,PARLIST=addr / (r)                }
```

JSWAKE=
Gives the address of the field containing the time specification for the timer event. The field must be aligned on a word boundary.

> **addr**
> Symbolic address (name) of the field.
>
> **(r)**
> Register containing the address value “addr”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses. (Address space ≤ 16 Mb.)

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses. (Address space ≤ 2 Gb.) Data lists start with the standard header.

PARLIST=
Gives the address of the operand list. The list should be aligned on a word boundary and must start with the standard header. The **DJSIPL** macro generates a description (DSECT/data list) of the operand list; the initialization values for the standard header are entered.

addr
Symbolic address (name) of the operand list.

(r)
Register containing the address value “addr”.

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:	0	0	a	a	a	a	a	a

A return code relating to the execution of the JSWAKE macro is transferred in register R15.

X'aaaaaa'	Meaning
X'000000'	Normal execution.
X'000008'	Operand error.
X'000010'	Unauthorized caller (not TSOS).

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

LDSLICE – Load slice

General

Application area: Linking and loading; see page 47
Macro type: Type S, MF format **2**: standard/C/D/L/E/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **LDSLICE** macro loads a slice defined in an LLM by the user into main memory. The slices used to build the LLM are defined by the user by means of the SET-USER-SLICE-POSITION statement (see the “BINDER” manual [5]).

Macro format and description of operands

```
LDSLICE
--------------------------------------------------------------

[ { MODULE=name          } ]
  { MODULE@=addr / (r)   }

[,MSG=*DBLOPT / INFORMATION / WARNING / ERROR / NONE ]

, { NAME=name          }
  { NAME@=addr / (r)   }

,PATH=NO / YES

,RELOAD=NO / YES

[,SLICE@=addr / (r) / label]

,MF=S / C / D / E / L / M

[,PARAM=addr / (r)]

,PREFIX=P / p

[,LABEL=name]
```

The operands are described in alphabetical order below.

LABEL=name
For MF=M only.
Name of the structure, i.e. the DSECT which describes the operand list of the **LDSLICE** macro. The operand is mandatory if there is no valid USING statement for the definition of the base address register for the DSECT of the parameter list. The LABEL operand must be specified in conjunction with the PARAM operand. Both operands are used to produce a valid USING statement.
The following may be specified for “name”:

1. The name specified in the name field of a preceding macro `name LDSLICE MF=D`.
2. The name “xSLICDS” if no “name” has already been specified, where “x” is the value of the PREFIX operand of a preceding macro `LDSLICE MF=D, PREFIX=x`.
   The default value for “x” is “P”.
3. The name of the longer DSECT containing the parameter list of the LDSLICE macro if the macro `LDSLICE MF=C` was specified earlier.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM and PREFIX, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form, D form or M form of the macro (see section “S-type macros” on page 29).

MODULE=name
Specifies the internal name of the LLM or PAM-LLM defined when the LLM was generated (max. 32 characters). The internal name should always be specified when several LLMs with user-defined slices are loaded in main memory. If the MODULE operand is not specified DBL selects the first LLM which contains the slice.

MODULE@=
For MF=M only.
Specifies the address of a field containing the internal name of the LLM.

addr
Symbolic address of the field containing the name.

(r)
Register containing the address value “addr”.

MSG=
Specifies the lowest message class; messages at and above this level will be output. The value set with the LOAD-EXECUTABLE-PROGRAM or START-EXECUTABLE-PROGRAM (or LOAD-PROGRAM or START-PROGRAM) load call is used as the default value.

***DBLOPT**
The operand value is taken from the last call of the MODIFY-DBL-DEFAULTS command. If a value for the parameter has not yet been set using the MODIFY-DBL-DEFAULTS command, MSG=INFORMATION applies.

INFORMATION
All classes of message will be output.

WARNING
Only messages of the WARNING and ERROR classes will be output. Messages of the INFORMATION message class will not be output.

ERROR
Only messages of the ERROR class will be output.

NONE
No messages will be output.

NAME=name
Specifies the name of the slice which is to be loaded. The name is specified by the user in the SET-USER-SLICE-POSITION statement when defining the slice. This name must be inserted for “name” (max. 32 characters long).
The specified slice can only be loaded if the root slice (%ROOT) is loaded.

NAME@=
For MF=M only.
Specifies the address of a field containing the name of the slice to be loaded.

addr
Symbolic address of the field containing the name.

(r)
Register containing the address value “addr”.

PATH=
Determines whether only the slice specified with NAME will be loaded or whether in addition to the NAMEd slice all “higher” slices in the same path (between the root slice and the NAMEd slice) will be loaded.

NO
Only the slice specified with NAME will be loaded.

YES
All higher slices in the same path will be loaded in addition to the NAMEd slice.

RELOAD=
Specifies whether a slice which is already loaded in main memory will be reloaded.

NO
An already loaded slice will not be reloaded.

YES
An already loaded slice will be reloaded and will overwrite the previous slice in main memory.

SLICE@=
Specifies the address of a 4-byte field in which DBL passes the load address of the slice. The address transferred refers to the first byte of the slice specified with NAME. The field must be aligned on a word boundary and have write access.
If the SLICE@ operand is not specified the load address of the slice is not passed.

addr
Symbolic address of the field. May be specified only if MF=M

(r)
Register containing the address value “addr”. May be specified only if MF=M.

label
Direct specification of the symbolic address of the field. May be specified only if MF=S or MF=L.

Rules for the loading of slices

- The slice to be loaded must form part of the physical structure for which the root slice was loaded.
- The root slice must already have been loaded using the START-PROGRAM or LOAD-PROGRAM command or the **BIND** macro.

Return information and error flags

If a field has been specified with the SLICE@ operand, DBL passes the load address of the slice.

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the LDSLICE macro is transferred in the standard header
(cc=subcode2, bb=subcode1, aaaa=main code):

X'cc'	**X'bb'**	**X'aaaa'**	**Meaning**
X'00'	X'00'	X'0000'	The macro was executed normally.
X'0C'	X'01'	X'0018'	A reserved field in the parameter list is not prefilled with zeros.
X'0C'	X'01'	X'0100'	Invalid parameter list. The mandatory NAME operand is missing.
X'0C'	X'20'	X'0102'	The specified module has not been found.
X'0C'	X'01'	X'0104'	The specified slice has not been found.
X'00'	X'00'	X'0108'	The specified slice has already been loaded.
X'0C'	X'01'	X'010C'	The field specified by SLICE@ is not aligned on a word boundary.
X'0C'	X'01'	X'0110'	The field specified by SLICE@ has read access only or is not assigned.
X'0C'	X'20'	X'0198'	Not enough memory space available for loading the object.
X'0C'	X'40'	X'0204'	Internal memory management error.
X'0C'	X'40'	X'0208'	Internal data manager error.
X'0C'	X'20'	X'0300'	Error during a system call.
X'00'	X'03'	X'FFFF'	The function is no longer or not yet supported.
X'00'	X'03'	X'FFFF'	The interface version is not supported.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

LEVCO – Modify priority level of contingency process

General

Application areas:	Contingency processing; see page 110 STXIT procedure; see page 131
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

Macro description

The **LEVCO** macro enables the user to modify the processing level (priority) of a current basic or contingency process. Permissible range:

Basic process: processing level 0 - 127 (default value: 0)
Contingency process: processing level 1 - 127 (default value: 1)

Macro format and description of operands

LEVCO
⎧ NEWLV=level / (r) ⎫ ⎩ NEWLVAD=addr / (r) ⎭ [,OLDLVAD=addr / (r)] ,QUEUE=FIFO / LIFO [,PARMOD=24 / 31] [,MF=L / (E,..)]

NEWLV=
Specifies the new processing level with which the process is to be continued.

> **level**
> Processing level (integer).
>
> **(r)**
> Register containing the specification for “level”.

NEWLVAD=
Specifies the new processing level with which the process is to be continued.

> **addr**
> Symbolic address of a 1-byte field containing the specification for the processing level.

(r)
Register containing the address value “addr”.

OLDLVAD=
Specifies the old processing level of the process that called **LEVCO**.

addr
Symbolic address of a 1-byte field in which the old processing level is to be entered.

(r)
Register containing the address value “addr”.

QUEUE=
Specifies the method by which the calling process with the modified processing level is to be enqueued with processes that will then have the same processing level.

FIFO
Indicates the “First In, First Out” method. This means that, after execution of the **LEVCO** macro, not only can the calling process be interrupted by processes with a higher processing level, but also that LIFO processes with the same processing level which have already been activated will be continued before it is started, since the calling process is placed at the end of the queue.

LIFO
Indicates the “Last In, First Out” method.
With its new processing level, the calling process is put at the head of the queue containing processes with the same processing level. Thus, it can be interrupted only by processes with a higher processing level or by LIFO processes with the same processing level.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses.
(Address space ≤ 16 Mb.)

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses.
(Address space ≤ 2 Gb.)

Notes on the macro call

A process must not reduce its processing level to a point where it is lower than that of a process already started and interrupted (see section “Contingency processes” on page 110). If the processing levels are equal, QUEUE=FIFO must not be specified (only QUEUE=LIFO is allowed).

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the macro is transferred in register R15.

X'aa'	X'bb'	Meaning
X'00'	X'00'	The process is continued with the new priority level.
X'04'	X'04'	No action. The new priority is lower than (or, in the case of FIFO, equal to) that of an already activated process.
X'10'	X'04'	Invalid operands were specified. No action.

LGOFF – Terminate job

General

Application area:	Starting, interrupting and terminating; see page 72
Macro type:	Type S, MF format **1**: 31-bit interface: standard/L/E/C/D form; see page 29

Macro description

The **LGOFF** macro allows the user to issue the EXIT-JOB command via the macro interface of the macro command language processor (MCLP) without interrupting program mode (see section “Macro Command Language Processor macros” on page 45). Messages concerning command processing are issued on SYSOUT.

The user ends a job with the EXIT-JOB command. Thereupon, the operating system deallocates the virtual memory pages and devices used by the job and prepares the output system files for output on printer or tape. If, during job execution, new file generations have been created, the system updates the base value of the file generation group (see the DMS manuals [7] and [8]). The name of the relevant file generation group, the first and the current generation and the base value are output in a message on SYSOUT.

Macro format and description of operands

```
LGOFF
----------------------------------------
    ⎧ 'BUT[,TAPE]'       ⎫
    ⎪ 'TAPE[,BUT]'       ⎪
  [ ⎨                    ⎬ ]
    ⎪ 'BUT[,NOSPOOL]'    ⎪
    ⎩ 'NOSPOOL[,BUT]'    ⎭

[,MF=C / D / L / (E,..)]
```

BUT=
This operand applies only to interactive users and is ignored in batch mode. It enables users to specify that they wish to start a new job after termination of the current one and that the connection to the server should therefore not be cleared down. If the BUT operand is omitted, a connection cleardown will be initiated.

TAPE=
This operand causes the system files to be spooled out to tape rather than to the printer. The SYSLST and/or SYSOUT files are written to a file on the same tape named “TAPE.TSNnnnn”, where nnnn is the TSN of the job to br terminated.
The SYSOPT file is written to a separate tape and is also given the file name “TAPE.TSNnnnn”, where nnnn is a new TSN of which the user is informed via the SYSOUT file.

NOSPOOL=
This operand prevents the system files SYSLST and SYSOUT (for LOGGING=PARAMETERS (LISTING=YES) in the SET-LOGON-PARAMETERS or MODIFY-JOB-OPTIONS commands) and SYSOPT from being output on the printer.

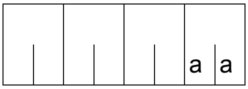
MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

Notes on the macro call

- The command operands must be enclosed in single quotes.
- An STXIT routine defined in the calling program for the “program termination” event class will be activated.
- Monitoring job variables (program and job) are set to '$T'.

Return information and error flags

R15:



A return code relating to the execution of the LGOFF macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Normal termination.
X'04'	The request has not been processed, due to insufficient memory space.
X'08'	Error in the operand list (address area).
X'0C'	The last output record entered in the user area has been truncated.
X'10'	Macro/command error (the command returned an error to MCLP).

Example

```
LGOFF    START
         PRINT NOGEN
         BALR  3,0
         USING *,3
         LGOFF 'NOSPOOL,BUT'
         TERM
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,lgoff), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,lgoff))
%  ASS6011 ASSEMBLY TIME: 408 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 77 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=lgoff
%  BLS0523 ELEMENT 'LGOFF', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'LGOFF', VERSION ' ' OF '<date> <time>' LOADED
/LOGOFF NOSPOOL,BUT
%  EXC0419 /LOGOFF AT 1505 ON <date> FOR TSN '52IS'
%  EXC0421 CPU TIME USED: 1.5734
%  JMS0150 INSTALLATION ' S200-40', BS2000 VERSION 'V190', HOST 'D016ZE04':
   PLEASE ENTER '/SET-LOGON-PARAMETERS' OR '?'
```

The output of system files was suppressed. The connection to the server is retained. A new SET-LOGON-PARAMETERS command may be entered immediately (see also the “Commands” manual [19], EXIT-JOB command).

LKCAN – Cancel lock request

General

Application area:	Distributed Lock Manager (DLM); see page 140
Macro type:	Type S, MF format **3**: C/D/L/M/E form; see page 29

Macro description

The **LKCAN** cancels lock requests which have not yet been allocated by the DLM. Lock requests which have been generated by the **LKENQ** macro and are in the WAITING or CONVERTING queue of the lock are completely canceled. If the lock request is in the GRANTED queue it is terminated with an error code.

Lock requests which are to be converted with the **LKCVT** macro are not canceled. Only the conversion job is canceled.

The call can be synchronous or asynchronous.

Macro formats and description of operands

LKCAN
MF=C / D / L / M / E ,ACKEVTT=*SYNCH / *TUCONTI / *TUEVENT / <var: enum-of _evttype_s:1> ,ACKNID=0 / <var: int:4> ,LOCKID=0 / <var: int:4> ,LSBADR=<var: pointer> ,USERPAR=0 / <var: int:4> ,PARAM=<var: pointer> / (reg: pointer>) ,PREFIX=N / p ,MACID=LDA / mac

The operands are described in alphabetical order below.

ACKEVTT=
Defines the way in which information about the cancellation is to be returned. There are three methods of control. The specified ID (contingency ID or event ID) is valid for the current cancellation request. Other cancellation requests from other tasks may specify different IDs.

<u>*SYNCH</u>
Synchronous lock request. Return from the macro if the lock request has been canceled or an error condition detected. The cancellation information is delivered in the return code.

***TUCONTI**
Contingency process. This value must be specified to obtain confirmation of the cancellation request during contingency processing

***TUEVENT**
Eventing. An event variable can be used to obtain the cancellation information. The cancellation information is supplied by calling the **SOLSIG** macro.

<var:enum-of _evttype_s:1>
Name of the field with the value of the cancellation method.

ACKNID=
For asynchronous lock requests, this operand specifies the contingency ID or the event ID which contains the information that the lock has now been deleted.

<var: int:4>
Default setting is 0.
Contingency ID or event ID.

LOCKID=
Lock ID for the lock request which is to be canceled.

<var: int:4>
Default setting is 0.
Lock ID which has been returned by the **LKENQ** macro.

LSBADR=
Field with the address of the Lock Status Block. The Lock Status Block contains the return code of the asynchronous call.

<var: pointer>
Name of the field with the address of the Lock Status Block.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID, and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

USERPAR=
The parameter for asynchronous messages which are to be transferred to the contingency process or to the event variable.

<var: int:4>
Default setting is 0. User-defined values.

Return information and error flags (return codes)

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the LKCAN macro is transferred in the standard header
(cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally.
X'00'	X'00'	X'0001'	The macro was executed normally. The cancel request has been enqueued. The lock request is canceled asynchronously.
X'00'	X'00'	X'0004'	Cancellation of the lock request was initiated, but the lock is still allocated.
X'00'	X'01'	X'1005'	The Lock Status Block is not accessible.
X'00'	X'01'	X'1006'	The ACKNID type is not the same as that specified in ACKEVTT.
X'00'	X'01'	X'100C'	The specification in the LOCKID operand is invalid.
X'00'	X'01'	X'1014'	The selected function is not allowed for the user.
X'00'	X'01'	X'10FF'	An incorrect parameter, which has no specific return code, was specified.
X'00'	X'20'	X'2001'	An internal error occurred.
X'00'	X'20'	X'2003'	Internal error in connection with the resource block.
X'00'	X'20'	X'2004'	Internal error in connection with a timeout.
X'00'	X'20'	X'2005'	Internal error in connection with the lock request.
X'00'	X'20'	X'2006'	Internal error in connection with XCS.
X'00'	X'82'	X'8004'	The lock has already been dequeued.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

LKCVT – Convert lock request

General

Application area:	Distributed Lock Manager (DLM); see page 140
Macro type:	Type S, MF format **3**: C/D/L/M/E form; see page 29

Macro description

The **LKCVT** macro converts an existing lock request. The lock must have been generated with the **LKENQ** macro and a lock ID must have been returned.

This lock ID (LOCKID) must be specified when the **LKCVT** macro is called. The user's lock request is identified by the lock ID.

The lock request is placed in the CONVERTING queue of the lock. If the lock can be allocated it is placed in the GRANTED queue. The **LKCVT** macro was executed successfully.

The lock request can be synchronous or asynchronous.

If it is asynchronous, the lock is allocated as soon as possible. The selected eventing is started and the allocation is notified to the calling program.

During allocation (conversion from a weaker to a stronger lock (LVBCTL=*MOVE).
During release (conversion from PW or EX mode to a weaker lock protection), the user can specify that the Lock Value Block is to be changed (LVBCTL=*MOVE).

Macro formats and description of operands

LKCVT
MF=C / D / L / M / E ,ASYNCTL=*SYNCH / *ASYNCH / <var: enum-of _asyncctl_s:1> ,CONTROL=*STD / *EXPRESS / <var: enum-of _ctltype_s:1> ,GRANTID=0 / <var: int:4> ,GRTEVTT=*SYNCH / *TUCONTI / *TUEVENT / <var: enum-of _evttype_s:1> ,HOLDTIM=*INFINITE / *SYSTEM / <var: int:2> / <integer 0..32767> ,LCKMODE=*NU / *CR / *CW / *PR / *PW / *EX / <var: enum-of _lckmode_s:1> ,LOCKID=0 / <var: int:4> ,LSBADR=<var: pointer> ,LVBCTL=*IGNORE / *MOVE / <var: enum-of _lvbctl_s:1>

LKCVT (cont.)

,RELEVTT=*NO / *TUCONTI / *TUEVENT / <var: enum-of _evttype_s:1>

,RELID=0 / <var: int:4>

,STATUS=*UNCHANGE / *RESET / *INVALIDATE / <var: enum-of _status_s:1>

,USERPAR=0 / <var: int:4>

,WAITTIM=*STD / *INFINITE / <var: int:2> / <integer 0..32767>

,WAITTYP=*TIME / *IMMEDIATE / <var: enum-of _timetype_s:1>

,PARAM=<var: pointer> / (reg: pointer>)

,PREFIX=N / p

,MACID=LDC / mac

The operands are described in alphabetical order below.

ASYNCTL=
Allocation control for an asynchronous lock request. If the lock request can be allocated immediately, a contingency process or an event signal can be suppressed.

***SYNCH**
If the lock request can be allocated immediately, a contingency process or an event signal is suppressed.

***ASYNCH**
Even if the lock request can be allocated immediately, a contingency process or an event signal is generated.

<var: enum-of _asyncctl_s:1>
Name of the field with the allocation control of the asynchronous lock request at the time of execution.

CONTROL=
Specifies the control options for the conversion request.

***STD**
Standard control of conversion request.

***EXPRESS**
This conversion request is to be executed before other conversion requests. This operand value should only be specified if the converted lock mode is released again very quickly.

<var: enum-of _ctltype_s:1>
Name of the field with the value which has been defined by the user.

GRANTID=
ID for the lock allocation. Specifies, for asynchronous lock requests, which contingency ID or event ID is to be assigned to the information that the lock has now been allocated.

<var: int:4>
Default setting is 0.
The contingency ID or event ID.

GRTEVTT=
Describes the way in which information about the lock allocation is to be returned. There are three methods of control: one synchronous and two asynchronous. The specified ID (contingency ID or event ID) is valid for the current lock request. Other lock requests from other tasks may specify different IDs.

***SYNCH**
Synchronous lock request. Return from the macro if the lock has been allocated or an error condition detected. The allocation information is delivered in the return code.

***TUCONTI**
Contingency process. This value must be specified to obtain confirmation of the allocation request during contingency processing. The user is notified of incompatible allocation requests which have been enqueued by other users. The release contingency is started if the lock is in an incompatible lock mode.

***TUEVENT**
Eventing. An event variable can be used to obtain allocation information. The allocation information is supplied by calling the **SOLSIG** macro. Incompatible allocation requests from other tasks are notified via the same event variable.

<var:enum-of _evttype_s:1>
Name of the field with the allocation event type at the time of execution

HOLDTIM=
The hold time is the time that the lock holder wants to hold the lock. When the hold time has expired, a release event is generated by the DLM so that the lock holder can weaken or release its lock request.
This can only be specified together with the RELEVTT operand.

***INFINITE**
No hold time limit.

***SYSTEM**
Hold time defined by the operating system.

<var: int:2>
Hold time defined by the user in seconds.

<integer 0..32767>
Direct specification of hold time in seconds.

LCKMODE=
Specifies the requested lock mode.

***NU**
The lock is in Null Mode and is always allocated. It is compatible with all other lock requests. However, access to the resource is not allowed.

***CR**
The lock is in Concurrent-Read Mode. Other locks are permitted only in Null Mode, Concurrent-Read Mode, Concurrent-Write Mode, Protected-Read Mode, and in Protected-Write Mode. The lock holder is granted unprotected read access to the resource.

***CW**
The lock is in Concurrent-Write Mode. Other locks are permitted only in Null Mode, Concurrent-Write Mode, or in Concurrent-Read Mode. The lock holder is granted unprotected write access to the resource.

***PR**
The lock is in Protected-Read Mode. Other locks are permitted only in Null Mode, Concurrent-Read Mode, or in Protected-Read Mode. The lock holder is granted protected read access to the resource.

***PW**
The lock is in Protected-Write Mode. Other locks are permitted only in Null Mode and in Concurrent-Read Mode. The lock holder is granted protected write access to the resource.

***EX**
The lock is in Exclusive Mode. Other locks are permitted only in Null Mode. Only the lock holder may access the resource.

<var: enum-of _lckmode_s:1>
Name of the field with the lock mode.

LOCKID=
Lock ID of the lock which is to be converted.

<var: int:4>
Default setting is 0.
Lock ID which has been returned by the **LKENQ** macro

LSBADR=
Field with the address of the Lock Status Block. The Lock Status Block contains the Lock Value Block.

<var: pointer>
Name of the field with the address of the Lock Status Block.

LVBCTL=
Control of the Lock Value Block.

*IGNORE
The Lock Value Block is not used.

***MOVE**
Read or write the Lock Value Block. The Lock Value Block can be read if the lock is allocated. It can be written if the lock is in PW or EX Mode.

<var: enum-of _lvbctl_s:1>
Name of the field with the control of the Lock Value Block at the time of execution.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID, and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form.

RELEVTT=
Defines the way in which information about the lock release is to be returned. There are two asynchronous methods of control. The specified ID (contingency ID or event ID) is valid for the current lock request.

_*NO_
No information about other incompatible lock requests concerning this lock is output. This can lead to deadlock problems if the lock is not released. If the hold time specified for the lock has expired, the lock holder for this lock request is terminated without further notification.

***TUCONTI**
Contingency process. This value must be specified to obtain confirmation of the release request during contingency processing. The user is notified of incompatible allocation requests which have been enqueued by other users. The release contingency is started if the lock is in an incompatible lock mode.

***TUEVENT**
Eventing. An event variable can be used to obtain the release information. The allocation information is supplied by calling the **SOLSIG** macro. Incompatible allocation requests from other tasks are notified via the same event variable.

<var:enum-of _evttype_s:1>
Name of the field with the release-event type at the time of execution.

RELID=
ID for the lock release. Specifies which contingency ID or event ID is to be assigned to the information that the lock request is blocking another lock request from a different user.

<var: int:4>
Default setting is 0.
The contingency ID or event ID.

STATUS=
Lock status change parameter. Specifies whether the lock status is validated when a lock is released in PW or EX Mode.

<u>*UNCHANGE</u>
The lock status is not changed while a lock is released in PW or EX Mode.

***RESET**
The lock status is reset to VALID while a lock is released in PW or EX mode.

***INVALIDATE**
The lock status is set to INVALID while a lock is released in PW or EX mode.

<var: enum-of _status_s:1>
Name of the field with the lock status change parameter at the time of execution.

USERPAR=
The parameter for asynchronous messages which are to be transferred to the contingency process or to eventing.

<var: int:4>
Default setting is 0.
User-defined values.

WAITTIM=
The wait time is the length of time it will take until the lock is allocated.

<u>*STD</u>
Standard wait time. If WAITTYP=*IMMEDIATE is specified the lock is allocated immediately or the job is terminated. If job processing is delayed (cluster, network) the wait time is 600 seconds.

***INFINITE**
The lock request never exceeds the time limit. The lock request waits until the lock is assigned or a deadlock situation is identified.

<var: int:2>
Wait time defined by the user in seconds.

<integer 0..32767>
Direct specification of wait time in seconds.

WAITTYP
Specifies the wait time type for waiting lock requests.

***TIME**
The wait time value is specified via the WAITTIM operand.

***IMMEDIATE**
Immediate lock request. The lock request is not enqueued if it cannot be allocated immediately. The value for the timeout can be specified via the WAITTIM operand. This value is used if the handling of the lock request leads to a DLM-internal wait status. This may occur, for example, if a network connection is no longer available after the lock request has been sent to a different node for processing, with the result that the necessary response is never transferred from that other node. To prevent this, the user can specify a wait time via the WAITTIM operand.

<var: enum-of _timetype_s:1>
User-defined values.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the LKCVT macro is transferred in the standard header
(cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally. The lock request was converted to a new lock mode.
X'00'	X'00'	X'0001'	The macro was executed normally. The lock request has been enqueued; the lock is allocated asynchronously.
X'00'	X'00'	X'0002'	The macro was executed. The lock was allocated in the specified lock mode, but the lock status is invalid. This information is output only if the lock mode is greater than or equal to CW.
X'00'	X'01'	X'1005'	The Lock Status Block is not available.
X'00'	X'01'	X'1006'	The GRANTID type is not the same as that specified in GRTTYPE.
X'00'	X'01'	X'1007'	The RELID type is not the same as that specified in RELTYPE.
X'00'	X'01'	X'1008'	Invalid combination of GRTEVTT and RELEVTT.
X'00'	X'01'	X'1009'	The specification in the WAITTIM operand is invalid.
X'00'	X'01'	X'100A'	The specification in the HOLDTIM operand is invalid
X'00'	X'01'	X'100C'	The specification in the LOCKID operand is invalid.
X'00'	X'01'	X'1015'	The specification in the LCKMODE operand is invalid.
X'00'	X'01'	X'10FF'	An incorrect parameter, which has no specific return code, was specified.

X'cc'	**X'bb'**	**X'aaaa'**	**Meaning**
X'00'	X'20'	X'2001'	An internal error occurred.
X'00'	X'20'	X'2003'	Internal error in connection with the resource block.
X'00'	X'20'	X'2004'	Internal error in connection with a timeout.
X'00'	X'20'	X'2005'	Internal error in connection with the lock request.
X'00'	X'20'	X'2006'	Internal error in connection with XCS.
X'00'	X'40'	X'4001'	A deadlock was detected.
X'00'	X'40'	X'4003'	The previous lock request has not yet been terminated.
X'00'	X'82'	X'8002'	Timeout.
X'00'	X'82'	X'8004'	The lock has already been dequeued.
X'00'	X'82'	X'8005'	The lock has already been canceled.
X'00'	X'82'	X'8006'	Immediate lock requests are not possible due to the existence of incompatible lock requests which have already been allocated.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

LKDEQ – Release lock request

General

Application area:	Distributed Lock Manager (DLM); see page 140
Macro type:	Type S, MF format **3**: C/D/L/M/E form; see page 29

Macro description

The **LKDEQ** releases a lock request from the queue of an existing lock. When the last (or only) lock request is released, the lock itself is canceled. The **LKDEQ** macro may be called synchronously or asynchronously. The lock is identified by its ID (LOCKID), which is returned by the **LKENQ** macro. This lock ID must be specified when the **LKDEQ** macro is called.
The user may specify that the Lock Value Block is to be changed while a lock is released in PW or EX Mode (LVBCTL=*MOVE).

Macro formats and description of operands

LKDEQ
MF=C / D / L / M / E ,ASYNCTL=*SYNCH / *ASYNCH / <var: enum-of _asyncctl_s:1> ,DEQEVTT=*SYNCH / *TUCONTI / *TUEVENT / <var: enum-of _evttype_s:1> ,DEQID=0 / <var: int:4> ,LOCKID=0 / <var: int:4> ,LSBADR=<var: pointer> ,LVBCTL=*IGNORE / *MOVE / <var: enum-of _lvbctl_s:l> ,STATUS=*UNCHANGE / *RESET / *INVALIDATE / <var: enum-of _status_s:1> ,USERPAR=0 / <var: int:4> ,PARAM=<var: pointer> / (reg: pointer>) ,PREFIX=N / p ,MACID=LDD / mac

The operands are described in alphabetical order below.

ASYNCTL=
Control of an asynchronous lock request. If the lock request can be dequeued immediately, a contingency process or an event signal can be suppressed.

<u>*SYNCH</u>
If the lock request can be released immediately, a contingency process or an event signal is suppressed.

***ASYNCH**
Even if the lock request can be released immediately, a contingency process or an event signal is generated.

<var: enum-of _asyncctl_s:1>
Name of the field with the asynchronous lock request at the time of execution.

DEQEVTT=
Describes the way in which release information is to be returned. There are three methods of control. The specified ID (contingency ID or event ID) is valid for the current release request. Other release requests from other tasks may specify different IDs.

<u>*SYNCH</u>
Synchronous lock request to dequeue the lock. Return from the macro if the lock has been released or an error condition detected. The release information is delivered in the return code.

***TUCONTI**
Contingency process. This value must be specified to obtain confirmation of the release request during contingency processing.

***TUEVENT**
Eventing. An event variable can be used to obtain the release information. The release information is supplied by calling the **SOLSIG** macro.

<var:enum-of _evttype_s:1>
Name of the field with the release information type at the time of execution.

DEQID=
ID for the lock request. Specifies, for asynchronous lock requests, which contingency ID or event ID is to be assigned to the information that the lock has now been released.

<var: int:4>
Default setting is 0.
The contingency ID or event ID.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID, and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form.

LOCKID=
Lock ID of the lock which is to be dequeued.

<var: int:4>
Default setting is 0.
Lock ID of a previous **LKENQ** macro call, which is to be dequeued.

LSBADR=
Field with the address of the Lock Status Block. The Lock Status Block contains the Lock Value Block.

<var: pointer>
Name of the field with the address of the Lock Status Block.

LVBCTL=
Control of the Lock Value Block.

<u>*IGNORE</u>
The Lock Value Block is not used.

***MOVE**
Read or write the Lock Value Block. The Lock Value Block can be read if the lock is allocated. It can be written if the lock is released in PW or EX Mode.

<var: enum-of _lvbctl_s:1>
Name of the field with the control of the Lock Value Block at the time of execution.

STATUS=
Specifies whether the lock status is validated when a lock is released in PW or EX Mode.

<u>*UNCHANGE</u>
The lock status is not changed while a lock is released in PW or EX Mode.

***RESET**
The lock status is reset to VALID while a lock is released in PW or EX mode.

***INVALIDATE**
The lock status is set to INVALID while a lock is released in PW or EX mode.

<var: enum-of _status_s:1>
Name of the field with the lock status change parameter at the time of execution.

USERPAR=
The parameter for asynchronous messages which are to be transferred to the contingency process or to eventing.

<var: int:4>
Default setting is 0.
User-defined values.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the LKDEQ macro is transferred in the standard header
(cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally. The lock was dequeued.
X'00'	X'00'	X'0001'	The macro was executed normally. The lock release request has been enqueued. The lock is dequeued asynchronously. The release is notified to the selected eventing methods.
X'00'	X'01'	X'1005'	The Lock Status Block is not available.
X'00'	X'01'	X'100C'	The specification in the LOCKID operand is invalid.
X'00'	X'01'	X'1006'	The DEQID type is not the same as that specified in DEQEVTT.
X'00'	X'01'	X'10FF'	An incorrect parameter, which has no specific return code, was specified.
X'00'	X'20'	X'2001'	An internal error occurred.
X'00'	X'20'	X'2003'	Internal error in connection with the resource block.
X'00'	X'20'	X'2004'	Internal error in connection with a timeout.
X'00'	X'20'	X'2005'	Internal error in connection with the lock request.
X'00'	X'20'	X'2006'	Internal error in connection with XCS.
X'00'	X'40'	X'4003'	The previous lock request has not yet been terminated.
X'00'	X'82'	X'8004'	The lock has already been dequeued.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

LKENQ – Generate lock

General

Application area:	Distributed Lock Manager (DLM); see page 140
Macro type:	Type S, MF format **3**: C/D/L/M/E form; see page 29

Macro description

The **LKENQ** macro generates a lock. The lock is not generated again if it already exists, but the calling task is allocated to it. The lock request is enqueued in the specified lock mode.

A lock is identified by a lock name and a scope. The lock name is supplied to the DLM by the user.

A lock ID (LOCKID) is returned to the calling task; this must be specified to facilitate processing in subsequent calls (**LKCVT**, **LKCAN**, **LKDEQ**). The lock ID is a task-specific control for the lock.

If the lock has already been enqueued for this task, the lock ID of the previous **LKENQ** request is returned but the macro is not executed. The lock mode has not been changed!

Macro formats and description of operands

LKENQ
MF=C / D / L / M / E ,ASYNCTL=*SYNCH / *ASYNCH / <var: enum-of _asyncctl_s:1> ,GRANTID=0 / <var: int:4> ,GRTEVTT=*SYNCH / *TUCONTI / *TUEVENT / <var: enum-of _evttype_s:1> ,HOLDTIM=*INFINITE / *SYSTEM / <var: int:2> / <integer 0..32767> ,LCKMODE=*NU / *CR / *CW / *PR / *PW / *EX / <var: enum-of _lckmode_s:1> ,LSBADR=<var: pointer> ,LVBCTL=*IGNORE / *MOVE / <var: enum-of _lvbctl_s:1> ,MULTENQ=*NO / *YES / <var: enum-of _multiple_s:1> ,NAMEADR=<var: pointer> ,NAMELEN=0 / <integer 8..48> / <var: int:2> ,NAMRNGE=*OWNSYSTEM / *CLUSTER / <var: enum-of _namerange_s:1> ,SCOPE=*NAMESPACEID / *USERID / *GROUPID / <var: emun-of_scope_s:1>

LKENQ (cont.)
,RELEVTT=*NO / *TUCONTI / *TUEVENT / <var: enum-of _evttype_s:1>
,RELID=0 / <var: int:4>
,TERMNTE=*STD / *FIRST / *SECOND / *THIRD / <var: enum-of _terminate_s:1>
,USERPAR=0 / <var: int:4>
,WAITTIM=*STD / *INFINITE / <var: int:2> / <integer 0..32767>
,WAITTYP=*TIME / *IMMEDIATE / <var: enum-of _timetype_s:1>
,PARAM=<var: pointer> / (reg: pointer>)
,PREFIX=N / p
,MACID=LDE / mac

The operands are described in alphabetical order below.

ASYNCTL=
Allocation control for an asynchronous lock request. If the lock request can be allocated immediately a contingency process or an event signal can be suppressed.

*SYNCH
If the lock request can be allocated immediately an allocation contingency or an event signal is suppressed.

*ASYNCH
An allocation contingency or an event signal is still generated by the DLM even if the lock request can be allocated immediately.

<var: enum-of _asyncctl_s:1>
Name of the field with the allocation control of the asynchronous lock request at the time of execution.

GRANTID=
ID for the lock allocation. Specifies, for asynchronous requests, which contingency ID or event ID is to be assigned to the information that the lock has now been allocated.

<var: int:4>
Default setting is 0.
The contingency ID or event ID.

GRTEVTT=
Describes the way in which information about the lock allocation is to be returned. There are three methods of control: one synchronous and two asynchronous. The specified ID (contingency ID or event ID) is valid for the current lock request. Other lock requests from other tasks may specify different IDs.

***SYNCH**
Synchronous lock request. Return from the macro if the lock has been allocated or an error condition detected. The information is delivered in the return code.

***TUCONTI**
Contingency process. This value must be specified to obtain confirmation of the allocation request during contingency processing. The user is notified of incompatible lock requests which have been enqueued by other users. The release contingency is started if the lock is in an incompatible lock mode.

***TUEVENT**
Eventing. An event variable can be used to obtain allocation information. The allocation information is supplied by calling the **SOLSIG** macro. Incompatible allocation requests from other tasks are notified via the same event variable.

<var:enum-of _evttype_s:1>
Name of the field with the allocation event type at the time of execution

HOLDTIM=
The hold time is the time that the lock holder wants to hold the lock. When the hold time has expired, a release event is generated by the DLM so that the lock holder can weaken or release its lock request.
This can only be specified together with the RELEVTT operand.

***INFINITE**
No hold time limit.

***SYSTEM**
Hold time defined by the operating system.

<var: int:2>
Hold time defined by the user in seconds.

<integer 0..32767>
Direct specification of hold time in seconds.

LCKMODE=
Specifies the lock mode of the requested lock.

<u>*NU</u>
The lock is in Null Mode and is always allocated. It is compatible with all other lock requests. However, access to the resource is not allowed.

***CR**
The lock is in Concurrent-Read Mode. Other locks are permitted only in Null Mode, Concurrent-Read Mode, Concurrent-Write Mode, Protected-Read Mode, and in Protected-Write Mode. The lock holder is granted unprotected read access to the resource.

***CW**
The lock is in Concurrent-Write Mode. Other locks are permitted only in Null Mode, Concurrent-Write Mode, or in Concurrent-Read Mode. The lock holder is granted unprotected write access to the resource.

***PR**
The lock is in Protected-Read Mode. Other locks are permitted only in Null Mode, Concurrent-Read Mode, or in Protected-Read Mode. The lock holder is granted protected read access to the resource.

***PW**
The lock is in Protected-Write Mode. Other locks are permitted only in Null Mode and in Concurrent-Read Mode. The lock holder is granted protected write access to the resource.

***EX**
The lock is in Exclusive Mode. Other locks are permitted only in Null Mode. Only the lock holder may access the resource.

<var: enum-of _lckmode_s:1>
Name of the field with the lock mode.

LSBADR=
Field with the address of the Lock Status Block. The Lock Value Block is part of the Lock Status Block.

<var: pointer>
Name of the field with the address of the Lock Status Block.

LVBCTL=
Specifies the handling of the Lock Value Block.

<u>*IGNORE</u>
The Lock Value Block is not used.

***MOVE**
Read or write the Lock Value Block. The Lock Value Block can be read if the lock is allocated. It can be written if the lock is released.

<var: enum-of _lvbctl_s:1>
Name of the field with the handling of the Lock Value Block at the time of execution.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID, and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form.

MULTENQ=
Multiple **LKENQ** requests are allowed.
Specifies whether the user is allowed to enqueue more than one lock request for the same lock name (= lock). Each of these lock requests is assigned its own lock ID (LOCKID) and each one must be dequeued individually.

<u>*NO</u>
If a lock request from this task already exists for the specified lock, this (the current) lock request is rejected.

***YES**
This (the current) lock request is enqueued irrespective of existing lock requests from this task for this lock. This lock request is assigned its own unique lock ID.

<var: enum-of _multiple_s:1>
Name of the field with the value of MULTENQ at the time of execution.

NAMEADR=
Field with the address of the lock name.

<var: pointer>
Name of the field with the address of the lock name.

NAMELEN=
Default setting is 0.
Specifies the length of the lock name.

<integer 8..48>
Direct specification of the length of the lock name at the time of transfer.

<var: int:2>
Name of the field with the length of the lock name at the time of execution.

NAMRNGE=
Specifies the range within which the lock name is valid. If a **LKENQ** request is executed on a single system (which is not and never will be part of a cluster), the lock is handled as if it were the single-system part of a cluster. This feature allows programs to be ported from single non-clustered systems to cluster systems.

<u>*OWNSYSTEM</u>
The specified lock name is valid on the local system only.

***CLUSTER**
The specified lock name is valid for the whole cluster.

<var: enum-of _namerange_s:1>
Name of the field with the range in which the lock name is valid, at the time of execution.

RELEVTT=
Describes the way in which information about the lock release is to be returned. There are three methods of control: one synchronous and two asynchronous. The specified ID (contingency ID or event ID) is valid for the current lock request. Other lock requests from other tasks may specify different IDs.

<u>*NO</u>
No information about other incompatible lock requests concerning this lock is output. This can lead to deadlock problems if the lock is not released. If the hold time specified for this lock has expired, the lock holder for this lock request is terminated without further notification.

***TUCONTI**
Contingency process. This value must be specified in order to obtain the release event during contingency processing. Users are informed of incompatible lock requests which are entered in the queue by other users. The release contingency is started if the lock is in an incompatible mode.

***TUEVENT**
Eventing. An event variable can be used to obtain the release event. The information is supplied when the **SOLSIG** macro is called. The incompatible lock requests from other tasks are reported via the same event variable.

<var:enum-of _evttype_s:1>
Name of the field with the release event type at the time of execution.

RELID=
ID for the lock release. Specifies which contingency ID or event ID is to be assigned to the information that the lock request is blocking another lock request from a different user.

<var: int:4>
Default setting is 0.
The contingency ID or event ID.

SCOPE=
Defines the local scope of the lock name.

<u>*NAMESPACEID</u>
Preset value: the specified lock name is used as the internal lock name. The first part (8 bytes) of the specified lock name implicitly forms the local scope. The local scope must be a character string. The valid characters are the letters “A..Z”, “a..z”; the digits “0..9” and the special characters “@” and “#”. The maximum length of the specified lock name is 48 characters.

***USERID**
The user ID to which the calling task belongs is used to form the internal lock name. The DLM determines the user ID and places it at the start of the lock name. The first part of the specified lock name is not interpreted as the local scope. The maximum length of the specified lock name is reduced to 40 characters.
If the operand SCOPE=*USERID is specified, an application's locks can be easily protected against access by another application. The applications simply have to be started under different user IDs.

***GROUPID**
The user group to which the calling task belongs is used to form the internal lock name. The DLM determines the user group and places it at the start of the lock name. The first part of the specified lock name is not interpreted as the local scope. The maximum length of the specified lock name is reduced to 40 characters.
The operand SCOPE=*GROUPID may only be specified if the software product SECOS is operational as otherwise the **LKENQ** call results in an error.

<var: enum-of _scope_s:1>
Name of the field with the local scope of the lock name at runtime.

TERMNTE=
Specifies the sequence in which this lock is released during abnormal termination of the lock-holder process (task or program). The termination sequence is divided into three classes. The locks of the process that is being terminated are released in the sequence specified, or they are all released simultaneously (as seen from the other processes).

<u>*STD</u>
This lock is released in the termination sequence specified by the DLM.

***FIRST**
This lock is released before or at the same time as the locks in the next class.

***SECOND**
This lock is released after or at the same time as the locks in the previous class, and before or at the same time as the locks in the next class.

***THIRD**
This lock is released after or at the same time as the locks in the previous class.

<var: enum-of _terminate_s:1>
Name of the field with the termination class at the time of execution.

USERPAR=
Contains the parameter for asynchronous messages. These are transferred to the contingency process or to eventing.

<var: int:4>
User-defined values. Default setting is 0.

WAITTIM=
The wait time is the length of time that must be waited until the lock is allocated.

<u>*STD</u>
Standard wait time. If WAITTYP=*TIME is specified, *INFINITA is assumed. If WAITTYP=*IMMEDIATE is specified the lock is allocated immediately or the job is terminated. If job processing is delayed (cluster, network) the wait time is 600 seconds.

***INFINITE**
Infinite wait time. The request never times out. The lock request waits until the lock is allocated or a deadlock situation is detected.

<var: int:2>
Wait time defined by the user in seconds.

<integer 0..32767>
Direct specification of wait time in seconds.

WAITTYP
Specifies the wait time type for waiting lock requests.

<u>*TIME</u>
The timeout value is specified via the WAITTIM operand.

***IMMEDIATE**
Immediate lock request. The lock request is not enqueued if it cannot be allocated immediately. The value for the timeout can be specified via the WAITTIM operand. This value is used if the handling of the lock request leads to a DLM-internal wait status. This may occur, for example, if a network connection is no longer available after the lock request has been sent to a different node for processing, with the result that the necessary response is never transferred from that other node. To prevent this, the user can specify a wait time via the WAITTIM operand.

<var: enum-of _timetype_s:1>
User-defined values.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the LKENQ macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally. The lock was allocated or the lock request has been enqueued.
X'00'	X'00'	X'0001'	The macro was executed normally. The lock request has been enqueued; the lock is allocated asynchronously.
X'00'	X'00'	X'0002'	The macro was executed but the lock status is invalid. This information is output only if the lock mode is greater than or equal to CW.
X'00'	X'01'	X'1001'	The lock name is too long.
X'00'	X'01'	X'1002'	The address of the lock name is not available.
X'00'	X'01'	X'1003'	The lock name is not allowed.
X'00'	X'01'	X'1004'	The specified name space is invalid.
X'00'	X'01'	X'1005'	The Lock Status Block is not available.
X'00'	X'01'	X'1006'	The GRANTID operand type is not the same as that specified in the GRTEVTT operand.
X'00'	X'01'	X'1007'	The RELID operand type is not the same as that specified in the RELEVTT operand.
X'00'	X'01'	X'1008'	Invalid combination of GRTEVTT and RELEVTT operands.
X'00'	X'01'	X'1009'	Invalid specification in the WAITTIM operand.

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'01'	X'100A'	Invalid specification in the HOLDTIM operand.
X'00'	X'01'	X'100B'	Invalid specification in the LCKTYPE operand.
X'00'	X'01'	X'1010'	The lock has already been enqueued for this task.
X'00'	X'01'	X'1015'	Invalid specification in the LCKMODE operand.
X'00'	X'01'	X'10FF'	An incorrect parameter, which has no specific return code, was specified.
X'00'	X'20'	X'2001'	An internal error occurred.
X'00'	X'20'	X'2003'	Internal error in connection with the resource block.
X'00'	X'20'	X'2004'	Internal error in connection with a timeout.
X'00'	X'20'	X'2005'	Internal error in connection with the lock request.
X'00'	X'20'	X'2006'	Internal error in connection with XCS.
X'00'	X'40'	X'4003'	The previous lock request has not yet been terminated.
X'00'	X'82'	X'8001'	No further locks can be created.
X'00'	X'82'	X'8002'	Timeout.
X'00'	X'82'	X'8003'	It is not possible at present to specify locks with NAMRNGE=*CLUSTER.
X'00'	X'82'	X'8005'	The lock request has already been canceled.
X'00'	X'82'	X'8006'	Immediate lock requests are not possible due to the existence of incompatible lock requests which have already been allocated.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

LKEQU – Generate DLM-specific layouts

General

Application area:	Distributed Lock Manager (DLM); see page 140
Macro type:	Definition macro, see page 28

Macro description

The **LKEQU** generates the DLM-specific layouts and values for the event type codes and the global return codes, which are set by the different macros of the DLM.

The **LKEQU** macro may only be called with MF = D.

Macro formats and description of operands

LKEQU
MF=D ,PARAM=<var: pointer> / (reg: pointer>) ,PREFIX=N / p ,MACID=LDQ / mac

Description of operands

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID, and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

Layout of the data area for LKEQU MF = D

```
          LKEQU MF=D
1         MFTST MF=D,PREFIX=N,MACID=LDQ,ALIGN=F,                       C
1               DMACID=LDQ,SUPPORT=(D),DNAME=LDQMDL
2 NLDQMDL  DSECT ,
2               *,##### PREFIX=N, MACID=LDQ #####
1 NLDQNMM# EQU   48                          maximum length of a lock name
1 *                                          including the name space id
1 *                                          which always must be the
1 *                                          first part of the lock name
1 *                                          string
1 *
1 NLDQNMP# EQU   48                          reserved
1 *
1 NLDQNSP# EQU   8                           length ot the name space id
1 *
1 NLDQTIMI EQU   -1                          value for infinite wait- or
1 *                                          holdtime
1 *
1 NLDQTIMS EQU   -2                          value for system defined
1 *                                          holdtime
1 *
1 NLDQUNIT EQU   247                         DLM unit number in standard
1 *                                          header
1 *
1 *   layout of the event data for asynchronous calls
1 *
1 NLDQREG3 DS    0XL4                        word 1 or register 3 of event
1 *                                          data
1 NLDQETC  DS    FL1                         unit which issued the event
1 *                                          (here always the event type
1 *                                          code of DLM)
1 *   event type code for event notification
1 NLDQVENT EQU   22                          DLM event
1 *
1 NLDQEVNT DS    FL1                         notified event
1 *   type of the event that occurred
1 NLDQGRT  EQU   1                           grant event after LKENQ or
1 *                                          LKCVT call
1 NLDQWTOT EQU   2                           waittimeout event after LKENQ
1 *                                          or LKCVT call
1 NLDQREL  EQU   3                           release event
1 NLDQHTOT EQU   4                           holdtimeout event
1 NLDQERGT EQU   5                           error event after failing
1 *                                          LKENQ or LKCVT call
1 NLDQCANC EQU   6                           cancelled event after LKENQ
1 *                                          or LKCVT call when a LKCAN
```

```
1 *                                        was issued meanwhile
1 NLDQLSBR EQU   7                         general error event when lock
1 *                                        state block was not
1 *                                        accessible
1 NLDQOKDQ EQU   8                         dequeue event after LKDEQ
1 *                                        call
1 NLDQERDQ EQU   9                         error event after failing
1 *                                        LKDEQ call
1 NLDQOKCN EQU   10                        cancel done event after LKCAN
1 *                                        call
1 NLDQERCN EQU   11                        error event after failing
1 *                                        LKCAN call
1 *
1 NLDQBMOD DS    X                         lock mode of the lock request
1 *                                        which is blocked by the own
1 *                                        granted lock request
1 NLDQFILL DS    X                         reserved
1 *
1 *
1 NLDQREG4 DS    0XL4                      word 2 or register 4 of event
1 *                                        data
1 NLDQUPAR DS    F                         last given value of user
1 *                                        parameter
1 *
1 NLDQEQUATES# EQU  *-NLDQETC
```

LKINF – Output information on locks

General

Application area:	Distributed Lock Manager (DLM); see page 140
Macro type:	Type S, MF format **3**: C/D/L/M/E form; see page 29

Macro description

The **LKINF** macro provides information on which locks are already being used. Several search filters can be activated to narrow the selection. The locks are identified by their lock names. The lock names may be fully or partially qualified names. If you specify partially qualified lock names, then, under certain circumstances, access may be very slow since the entire DLM database has to be searched for hits.

Macro formats and description of operands

LKINF
MF=C / D / L / M / E
,CONTROL=list-poss(3): *LOCKED / *LOCKED_BY_ME / *LCKMODE
,LCKMODE=*NU / *CR / *CW / *PR / *PW / *EX / <var: enum-of _lckmode_s:1>
,NAMEADR=<var: pointer>
,NAMELEN=0 / <integer 0..48> / <var: int:2>
,NAMRNGE=*OWNSYSTEM / *CLUSTER / <var: enum-of _namerange_s:1>
,SCOPE=*NAMESPACEID / *USERID / *GROUPID / <var: emun-of_scope_s:1>
,NAMTYPE=*FULL / *PARTIAL / <var: enum-of _nametype_s:1>
,PARAM=<var: pointer> / (reg: pointer>)
,PREFIX=N / p
,MACID=LDI / mac

The operands are described in alphabetical order below.

CONTROL=
The search filter controls which data is to be returned.

> **list-poss(3):**
> A list containing a maximum of four elements can be formed from the following values.
>
> ***LOCKED**
> Only information about locks with a lock mode not equal to zero is returned.

***LOCKED_BY_ME**
Only information about locks which the user is maintaining in a lock mode not equal to zero is returned. The user is identified by his or her lock ID (LOCKID).

***LCKMODE**
Only information relating to the specified lock mode is returned.

LCKMODE=
Specifies the requested lock mode. Is only evaluated if the operand CONTROL=*LCKMODE is also specified.

<u>*NU</u>
The lock is in Null Mode and is always allocated. It is compatible with all other lock requests. Access to the resource is not allowed.

***CR**
The lock is in Concurrent-Read Mode. Other locks are permitted only in Null Mode, Concurrent-Read Mode, Concurrent-Write Mode, Protected-Read Mode, and in Protected-Write Mode. The lock holder is granted unprotected read access to the resource.

***CW**
The lock is in Concurrent-Write Mode. Other locks are permitted only in Null Mode, Concurrent-Write Mode, or in Concurrent-Read Mode. The lock holder is granted unprotected write access to the resource.

***PR**
The lock is in Protected-Read Mode. Other locks are permitted only in Null Mode, Concurrent-Read Mode, or in Protected-Read Mode. The lock holder is granted protected read access to the resource.

***PW**
The lock is in Protected-Write Mode. Other locks are permitted only in Null Mode and in Concurrent-Read Mode. The lock holder is granted protected write access to the resource.

***EX**
The lock is in Exclusive Mode. Other locks are permitted only in Null Mode. Only the lock holder may access the resource.

<var: enum-of _lckmode_s:1>
Name of the field with the lock mode.

Note

The LCKMODE and CONTROL operands may not be specified together. The lock mode may only be specified directly via the LCKMODE operand or indirectly via the CONTROL operand.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID, and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form.

NAMEADR=
Field with the address of the lock name.

<var: pointer>
Name of the field with the address of the lock name.

NAMELEN=
Default setting is 0.
Specifies the length of the lock name.

<integer 0..48>
Direct specification of the length of the lock name at the time of transfer.

<var: int:2>
Name of the field with the length of the lock name at the time of execution.

NAMRNGE=
Specifies the range within which the lock name is valid.

***OWNSYSTEM**
The specified lock name is valid on the local system only.

***CLUSTER**
The specified lock name is valid for the whole cluster.

<var: enum-of _namerange_s:1>
Name of the field with the range in which the lock name is valid.

NAMTYPE=
Specifies whether the character string for the name is a fully or partially qualified lock name.

***FULL**
The specified lock name is a fully qualified lock name.

***PARTIAL**
The specified lock name is a partially qualified lock name.

<var: enum-of _nametype_s:1>
Name of the field with the lock name type.

SCOPE=
Determines the local scope of the lock name

<u>*NAMESPACEID</u>
Preset value: the specified lock name is used as the internal lock name. The first part (8 bytes) of the specified lock name implicitly forms the local scope. The local scope must be a character string. The valid characters are the letters “A..Z”, “a..z”; the digits “0..9” and the special characters “@” and “#”. The maximum length of the specified lock name is 48 characters.

***USERID**
The user ID to which the calling task belongs is used to form the internal lock name. The DLM determines the user ID and places it at the start of the lock name. The first part of the specified lock name is not interpreted as the local scope. The maximum length of the specified lock name is reduced to 40 characters.
If the operand SCOPE=*USERID is specified, an application's locks can be easily protected against access by another application. The applications simply have to be started under different user IDs.

***GROUPID**
The user group to which the calling task belongs is used to form the internal lock name. The DLM determines the user group and places it at the start of the lock name. The first part of the specified lock name is not interpreted as the local scope. The maximum length of the specified lock name is reduced to 40 characters.
The operand SCOPE=*GROUPID may only be specified if the software product SECOS has been purchased and is operational as otherwise the **LKENQ** call results in an error.

<var: enum-of _scope_s:1>
Name of the field with the local scope of the lock name at runtime.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the LKINF macro is transferred in the standard header
(cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally. At least one corresponding lock was found.
X'01'	X'00'	X'0000'	The macro was executed but no corresponding lock was found.
X'02'	X'00'	X'0000'	The macro was executed and corresponding locks were found but not all the nodes returned information as a result of a network problem.
X'00'	X'01'	X'1001'	The lock name is too long.
X'00'	X'01'	X'1002'	The address of the lock name is not available.
X'00'	X'01'	X'1003'	The specified lock name is not permissible.
X'00'	X'01'	X'1004'	The specified name space is invalid.
X'00'	X'01'	X'100F'	The search filter is not permissible.
X'00'	X'01'	X'1015'	The specification in the LCKMODE operand is invalid.
X'00'	X'01'	X'1016'	The specification in the NAMTYPE operand is invalid.
X'00'	X'01'	X'1017'	The specification in the NAMRNGE operand is invalid.
X'00'	X'01'	X'10FF'	An incorrect parameter, which has no specific return code, was specified.
X'00'	X'20'	X'2001'	An internal error occurred.
X'00'	X'20'	X'2003'	Internal error in connection with the resource block.
X'00'	X'20'	X'2004'	Internal error in connection with a timeout.
X'00'	X'20'	X'2005'	Internal error in connection with the lock request.
X'00'	X'20'	X'2006'	Internal error in connection with XCS.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

LKLSB – Generate Lock Status Block layout

General

Application area:	Distributed Lock Manager (DLM); see page 140
Macro type:	Type S, MF format **3**: C/D/L/M/E form; see page 29

Macro description

The **LKLSB** macro generates the layout of the Lock Status Blocks (LSB). The LSB is part of the user address space which must be generated by the user for an asynchronous lock request. This user address space must be accessible for the DLM until the generated lock request is terminated.

Termination of the asynchronous lock request causes a corresponding return code to be written to the LSB. The user-defined event method for this lock request is started.

To decide whether the return code in the LSB was delivered by the DLM, the user must initialize the LSB by initializing the return code with the return code value when calling `LKLSB MF=L`.

Macro formats and description of operands

LKLSB
MF=C / D / L / M ,PARAM=<var: pointer> / (reg: pointer>) ,PREFIX=N / p ,MACID=LDL / mac

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID, and PARAM), see page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form.

Initialization of the Lock Status Block

```
           LKLSB MF=L
1          MFTST MF=L,PREFIX=N,MACID=LDL,ALIGN=F,                             C
1                DMACID=LDL,SUPPORT=(D,C,M,L)
2          DS    OF
1          FHDR  MF=L,UNIT=247,FUNCT=6,VERS=1,RC=-1
2          DS    OA
2          DS    OXL8                GENERAL OPERAND LIST HEADER
2          DC    AL2(247)            FUNCTION UNIT NUMBER
2          DC    AL1(6)              FUNCTION NUMBER
2          DC    AL1(1)              FUNCTION INTERFACE VERSION NUMBER
2          DC    A(-1)                Returncode
1 *                                       FHDR
1          DC    A(0)                     LOID
1          DC    A(0)                     OWID
1          DC    CL16' '                  VAL
```

LPOV – Load segment

General

Application area:	Linking and loading; see page 47
Macro type:	Type O; see page 28

Macro description

The **LPOV** macro (Load Program OVerlay) allows only the (one) specified segment to be loaded into memory, even if it is already in memory.
The **LPOV** macro is used if nonautomatic segment loading is required. Automatic segment loading permits more than one segment to be loaded with one macro (see the “Utility Routines” manual [27]).

Macro format and description of operands

LPOV
,modulename [,address]

modulename
Specifies the symbolic name of the segment to be loaded. This name can be up to 6 alphanumeric characters long.

address
Specifies the symbolic address in the calling or another module, to which control is returned after the module has been loaded. If this operand is omitted, control is returned to the instruction following the **LPOV** macro. The calling program waits until the module is loaded.

Functional description

Execution of the **LPOV** macro causes the static loader to be invoked; this loads the segment specified by “modulename” into memory. After the loading process, control is passed to the instruction specified by “address”. This address may be contained in the calling module or be an external reference in another module. If the “address” operand is omitted, control is passed to the instruction following the **LPOV** macro.
If an external reference is used for control transfer, it is the user's responsibility to ensure that the module with the relevant entry address is in memory after the loading process.
Since execution of the **LPOV** macro causes only one segment to be loaded, each segment must be loaded explicitly. When the **LPOV** macro is executed, the segment specified is loaded irrespective of whether it is in memory or not. The **LPOV** macro is not linked to the overlay control module, and no list of overlay segments currently in memory is maintained.

Notes on the macro call

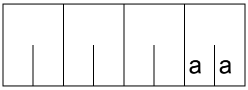
– After a segment has been loaded by the **LPOV** macro, corrections may be made using AID with `%ON %LPOV` (see the “AID” manual [3]).

– The **LPOV** macro, which permits the nonautomatic loading of segments, communicates directly with the operating system. If **LPOV** is used concurrently with the **CALL** or **SEGLD** macro (automatic segment loading), the status list of the program overlay structure may be corrupted (see the “Utility Routines” manual [27]). Such a condition may lead to errors during program execution.

Return information and error flags

If errors occur during loading of an overlay segment, the Executive returns control to the program at the instruction following the **LPOV** macro expansion. If no errors occur, the continuation address specified in the macro call is valid.

R15:



A return code relating to the execution of the LPOV macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	The load operation was successful.
X'04'	Invalid read-only modification record.
X'08'	Illegal record code in text/modification block or a modification record precedes the first text record.
X'0C'	There are too many read-only modification records or the segemt name cannot be found in any of the index records for load modules (=segments).
X'10'	There is no index record for load modules having the segment name %ROOT.
X'14'	There is not sufficient memory to load the segment.
X'18'	Illegal key information, error during reading of PAM load module file or error in the output of messages.
X'1C'	Invalid segment name.
X'20'	AID load error.
X'24'	Error in reading C element from PLAM library.

MINF – Output memory map for class 6 memory or memory pool

General

Application areas:	Working with virtual memory; see page 55 Memory pools; see page 55 Requesting and accessing lists and tables; see page 155
Macro type:	Type S, MF format **1**: 31-bit interface: standard/L/D/E form; see page 29

The size of class 6 memory may vary within a program run, depending on class 5 and class 6 memory space requirements.

Macro description

The **MINF** macro provides the user with information about the utilization of

- the class 6 memory allocated to the user, or
- a memory pool in class 6 memory.

The information is output in the form of a bit table which indicates for a particular memory page whether it is occupied or not. The entries in the bit table have the following meaning:

byte 0, bit 2^7 represents the 1st (requested) memory page,
byte 0, bit 2^6 represents the 2nd memory page,
:
:
byte 0, bit 2^0 represents the 8th memory page,
byte 1, bit 2^7 represents the 9th memory page,
etc.

Memory page utilization is indicated thus:

bit 2^n = 0: memory page is occupied.
bit 2^n = 1: memory page is not occupied.

Note

> If information on class 6 memory is to be output (CL6 operand), each page within a memory pool is considered to be occupied, regardless of whether or not the page has already been requested by means of **REQMP**.
> If information on the memory pool itself is to be output (MP operand), only those pages that have already been requested by means of **REQMP** are considered to be occupied.

Macro format and description of operands

```
MINF

 ⎧ CL6 ⎫
 ⎩ MP  ⎭

,INF=⎧ SIZE          ⎫
     ⎩ FREE,MAP=addr ⎭

,ADDR=addr

,MF=S / (E,...) / L / (D,pre) / D
```

CL6
Information about the utilization of the class 6 memory allocated to the user is output. Pages of a memory pool are always marked as in use.

MP
Information about the utilization of a memory pool (class 6 memory) is output. Only those pages are marked as in use which have been requested by means of **REQMP**.

INF=
Specifies the type of information requested.

> **SIZE**
> The virtual page number (VPN) of the first page and the size (number of pages) of the class 6 memory/memory pool are output. The information is output to the field specified by the ADDR operand.
>
> **FREE,MAP=addr**
> A bit table is output, indicating the utilization of pages of the memory area specified with the ADDR operand. The bit table is output to the field specified by the MAP operand. Additional information about the length of the bit table is entered in the field specified with ADDR.
> “addr“ is the symbolic address (name) of a field (1 byte of the field's length is required for every 8 memory pages).

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A prefix (pre = 1..3 letters) can be specified in the D form of the macro, as shown in the macro format.
Default setting: pre = MNF

ADDR=addr
Specifies a field which is used for both input and output (operand INF).
Field length = 16 bytes (4 words). The field must be aligned on a word boundary.

addr
Symbolic address (name) of the field.

The contents depend on the specification for CL6/MP/SIZE, as illustrated below:

CL6, INF=SIZE: (output field only)

Byte	Output
0 - 3	VPN of first memory page below 16 Mb
4 - 7	Number of memory pages below 16 Mb
8 - 11	VPN of first memory page above 16 Mb or X' 00000000'
12 - 15	Number of memory pages above 16 Mb or X' 00000000'

X'00000000' is output if no class 6 memory exists above 16 Mb.

MP,INF=SIZE:

Byte	Input	Output
0 - 3	VPN of any pool page	VPN of 1st memory pool page
4 - 7	not used	size (number of pages) of memory pool
8 - 11	not used	not changed [1]
12 - 15	not used	not changed [1]

[1] 'not changed' means: In the event of a number of consecutive MINF calls, the contents specified with the preceding call are retained.

CL6/MP,INF=FREE:

Byte	Input	Output
0 - 3	VPN of 1st page of requested area	not changed
4 - 7	Number of memory pages about whose utilization a bit table is to be output	Number of memory pages actually described by the bit table
8 - 11	not used	not changed
12 - 15	not used	not changed

The specified VPN must be a multiple of 16 (n=0,1,...)

Return information and error flags

After macro processing, register R1 contains the operand list address.

R15:

0	0	0	0	0	0	a	a

A return code relating to the execution of the MINF macro is transferred in register R15. The same RC is additionally transferred in the standard header of the operand list.

X'aa'	Meaning
X'00'	Function has been executed
X'04'	Operand error (illegal address specified with MF=(E,...)
X'08'	Invalid VPN (the specified page is not located in class 6 memory/memory pool, or the value is not a multiple of 16 (n = 0, 1, ...)
X'0C'	Address error (operand list/ADDR field/MAP)

Example

In the example, information about the size, location and page utilization of a memory pool is requested; the memory pool has been created above the 16-Mb boundary.

```
  MINF     START
           PRINT NOGEN
  MINF     AMODE 31
  MINF     RMODE ANY
           GPARMOD 31
1                 *,MACRO: GPARMOD, VERSION: VER121
           BALR  3,0
           USING *,3
           ENAMP MPNAME=MEMP,SCOPE=GLOBAL,MPIDRET=PID,BSIZE=48 ---------  (1)
1                *,ENAMP: 144/951025
  REQMP    REQMP MPID=PID,BSIZE=5 ------------------------------------  (2)
1                *,REQMP: 141 / 950210
  *
  DTH1     LR    4,1 -------------------------------------------------  (3)
           LA    4,4095(1)
           LA    4,1(4)
           LR    5,4
           SRL   5,12
           ST    5,MPINF1
  MINF1    MINF  MP,ADDR=MPINF,INF=SIZE ------------------------------  (4)
1                 *,MACRO: MINF, VERSION: VER174
  DTH2     MVC   MPINF2,=F'16'
           MVC   0(27,4),TEXT
  MINF2    MINF  MP,ADDR=MPINF,INF=FREE,MAP=BITTAB --------------------  (5)
1                 *,MACRO: MINF, VERSION: VER174
```

```
TERM     TERM
*
****  Definitions  ****
*
MPINF    DS    0F
MPINF1   DS    F
MPINF2   DS    F
MPINF3   DS    F
MPINF4   DS    F
PID      DS    F
BITTAB   DS    CL4
TEXT     DC    C'2ND PAGE OF MEMORY POOL'
         END
               =F'16'
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,minf), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,minf)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 358 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 82 MSEC
//end
%  ASS6012 END OF ASSEMBH
/load-executable-program library=macexmp.lib,element-or-symbol=minf, -
/     test-options=*aid,prog-mode=*any
%  BLS0523 ELEMENT 'MINF', VERSION '@' FROM LIBRARY
   ':20SG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'MINF', VERSION ' ' OF '<date> <time>' LOADED
/%in reqmp <%d c' ',c'- label reqmp:',%1,%15>
/%in dth1 <%d c' ',c'- label dth1:',%1,%15>
/%in minf1 <%d c' ',c'- label minf1:',mpinf1, mpinf1 %x>
/%in dth2 <%d c' ',c'- label dth2:',mpinf1, mpinf1 %x,mpinf2, mpinf2 %x>
/%in term <%d c' ',c'- label term:',%@(bittab) -> %xl4;%d %4 -> %xl30>
/%r
```

```
- LABEL REQMP: ------------------------------------------------------------- (6)
*** TID: 005000D8 *** TSN: 2QSE *********************************************
CURRENT PC: 01000040    CSECT: MINF  ****************************************
**
%1              = 01100000
%15             = 04000000

- LABEL DTH1: -------------------------------------------------------------- (7)
CURRENT PC: 0100006A    CSECT: MINF  ****************************************
%1              = 01100000
%15             = 00000000

- LABEL MINF1: ------------------------------------------------------------- (8)
SRC_REF:   100 SOURCE: MINF  PROC: MINF  ************************************
MPINF1          =        4353
CURRENT PC: 01000080    CSECT: MINF  ****************************************
V'010000E0' = MPINF1   + #'00000000'
010000E0 (00000000) 00001101                                 ....

- LABEL DTH2: -------------------------------------------------------------- (9)
SRC_REF:   117 SOURCE: MINF  PROC: MINF  ************************************
MPINF1          =        4352
CURRENT PC: 0100009A    CSECT: MINF  ****************************************
V'010000E0' = MPINF1   + #'00000000'
010000E0 (00000000) 00001100                                 ....
SRC_REF:   117 SOURCE: MINF  PROC: MINF  ************************************
MPINF2          =         256
CURRENT PC: 0100009A    CSECT: MINF  ****************************************
V'010000E4' = MPINF2   + #'00000000'
010000E4 (00000000) 00000100                                 ....

- LABEL TERM: ------------------------------------------------------------- (10)
CURRENT PC: 010000C2    CSECT: MINF  ****************************************
V'010000F4' = MINF     + #'000000F4'
010000F4 (000000F4) 07FF0000                                 .~..
V'01101000' = ABSOLUT  + #'01101000'
01101000 (01101000) F2D5C440 D7C1C7C5 40D6C640 D4C5D4D6    2ND PAGE OF MEMO
01101010 (01101010) D9E840D7 D6D6D300 00000000 0000        RY POOL.......
```

(1) Create a memory pool above the 16-Mb boundary.
Size of the memory pool = 48 memory pages (rounded to 1 Mb).

(2) Reserve 5 memory pages beginning with the start address.

(3) REQMP supplies the start address of the memory pool in register R1. The field `MPINF1` is the input field for the macro **MINF** and as such should contain (any) virtual page number (VPN) within the memory pool. Register R4 contains the address of the first byte on the second memory pool page, while register R5 contains the relevant VPN. This VPN is stored in the field `MPINF1`.

(4) First **MINF** call:
Output size and location of the memory pool.
Output is to the field `MPINF`: The VPN of the first page of the memory pool is in the field `MPINF1`, and the number of pages of the memory pool is in the field `MPINF2`. The fields `MPINF3` and `MPINF4` are not modified.

(5) Second **MINF** call:
A bit table of the first 16 pages of the memory pool is to be output (the preceding MVC transfers the figure 16 into the second input field `MPINF2`).
The input field for **MINF** now contains the VPN of the first page (`MPINF1`) supplied by the first **MINF** call and the number of pool pages whose utilization is to be output (`MPINF2`). Output is to the field `BITTAB`.

(6) After execution of **ENAMP**, the start address and the return code are interrogated. The memory pool starts at address X'01100000'.
RC = X'04000000' means: a new memory pool has been created.

(7) After execution of **REQMP**, the address of the reserved area and the return code are interrogated. The reserved area starts at address X'01100000';
RC = X'00000000'.

(8) The input field for **MINF** contains X'00001101' = 4353 as the VPN of a memory pool page.

(9) After the first **MINF**:
The VPN of the first page and the size of the memory pool are output:
VPN = 4352 (≙X'1100'); size = 256 pages (≙X'100'). The VPN of the first page is retained as input for the second **MINF** call.

(10) After the second **MINF**:

- The bit table starts at address X'000000EC'; length = 2 bytes (≙ 16 pages). Contents: X'07FF'; bit pattern: 0000/0111/1111/1111 . The memory pages reserved with **REQMP** are marked as in use; the remainder of the 16 pages is marked as free.
- The text `2ND PAGE ...` has been entered in the second page of the memory pool.

MSG7X – Output message

General

Application area:	Messages; see page 161
Macro type:	Type S, MF format **3**: C/D/L/M/E form; see page 29

Register contents may be specified only in the M form of the macro.

Macro description

The **MSG7X** macro outputs a system message to SYSOUT, SYSLST, the operator console, a user program area or an S variable. The specified message code must contain 7 characters.
The **MSG7X** macro uses the new operand list layout (with standard headers; see page 43).

Every system message has a 7-character message code. The first 3 characters of the code denote the message class; the remaining 4 characters are used for consecutive numbering within a class. System messages can contain variable sections “(&nnn)” that can be replaced by inserts.

Macro format and description of operands

```
MSG7X
--------------------------------------------------------------------------
ID=msgid / (r1) / (class,(r)) / (,(r)) / (class,(addr)) / (,addr)

              ⎧                  ⎧ addr     ⎫        ⎫
              ⎪ (insertlength,   ⎨ (r)      ⎬)       ⎪
              ⎪                  ⎩ 'insert' ⎭        ⎪
              ⎪                                      ⎪
              ⎪                   ⎧ addr     ⎫       ⎪
[,INSERT=     ⎨ ((insertlength,   ⎨ (r)      ⎬),...) ⎬]
              ⎪                   ⎩ 'insert' ⎭       ⎪
              ⎪ ((r1),(r2))                          ⎪
              ⎪ (((r1),(r2)),...)                    ⎪
              ⎪ number                               ⎪
              ⎩ NONE                                 ⎭
[,LAN='language']

,DEST=SYSOUT / SYSLST / CONSOLE / (destination,...) / NONE

[,UCDEST='destcode' / destaddr / (r) / N]
```

MSG7X (cont.)

```
            ⎧                             ⎫
            ⎪ (replylength,⎧ addr ⎫)      ⎪
[,REPLY=    ⎨              ⎩ (r)  ⎭       ⎬]
            ⎪ ((r1),(r2))                 ⎪
            ⎩ N                           ⎭

            ⎧                             ⎫
            ⎪ (bufferlength,⎧ addr ⎫)     ⎪
[,BUFFER=   ⎨               ⎩ (r)  ⎭      ⎬]  ,MAP=NO / YES
            ⎪ ((r1),(r2))                 ⎪
            ⎩ N                           ⎭

,DMS=APPL / NOTAPPL / NA

,BUFFUSE=INTERNAL / EXTERNAL

,DEFTEXT=NONE / (textlength,addr) / (textlength,(r)) / ((r1),(r2))

,TIMER=UNLIMITED / (r) / wert

,TIMESTAMP=NO / YES

,DATESTAMP=NO / YES

,MF=D / C / L / E / M

[,PARAM=addr / (r)]

,PREFIX=X / p

,MACID=MSG / macid
```

ID=

Specifies the message code of the system message to be output.

msgid

Specifies a 7-character message code.

(class,addr)

Modifies the message code in the parameter list.

(class,(r))

This operand value may be specified only in conjunction with MF=M.

`class` message class

`addr` symbolic address (name) of a field containing the new message number

`(r)` register containing the new message number

(,addr)

Modifies the message number in the parameter list.

(,(r))

This operand value may be specified only in conjunction with MF=M.

addr	symbolic address (name) of a field containing the new message number
(r)	register containing the new message number

INSERT=

Specifies up to 30 lengths and addresses of inserts. An address reference is created in the message processing operand list for each entry.
If more inserts are specified than can be accommodated by the message, excess inserts are ignored. An insert consisting solely of blanks is shortened to a single blank.
Trailing blanks in an insert are suppressed. The character X'01' must be entered at the end of an insert to prevent the suppression of blanks. When the character X'01' is encountered during message processing, all blanks that precede it are retained.
If insertlength=0 and addr=0 are specified, the system assumes the default value from the message file for the insert. If no default value is specified in the message file the insert is omitted.
If a message text contains more inserts than are specified in the macro call, the default value is assumed for every excess insert. The substitute insert (&nn) is used if no default value is present.
If MF=M is specified, an address reference is created for all inserts specified for INSERT. For example, calling MF=M,INSERT=(((R1),(R2)),((R3),(R4))) changes inserts 0 and 1 in the data area but not 2 or 3.
All inserts defined previously in the operand list are ignored. The number of inserts defined for MF=M must be less than or equal to any maximum number predefined by MF=C or MF=D.

(insertlength,...)

insertlength	length of the insert
addr1	symbolic address (name) of the area with the insert
(r)	register containing the insert address (may be specified only if MF=M)
'insert'	direct specification of the insert (length of string: max. 4 characters)

If insertlength=0, the insert must begin with a record length field (4 bytes) and is thus always at least 4 bytes long.

Byte 0-1	length of the insert
Byte 2-3	reserved.

Inserts may be skipped by entering commas for omitted positions, e.g.
INSERT=(,(insertlength2,addr2),,(insertlength4,addr4)). Inserts omitted in this way are replaced by their default values or by a blank character string (&nn).
The sum of the insertion lengths must be ≤ 4079 bytes. An insertion list must be enclosed within additional paretheses.
If output is to console, the sum of all insertions must be ≤ 218 bytes.

((r1),(r2))

`r1` register containing insert length
`r2` register containing insert address.
This may only be specified with MF=M.
An insertion list must be enclosed within additional paretheses.

number
Number of INSERTs for which space is to be reserved in the data area for message processing.

With MF=C/D, the number is 15 by default.
With MF=M, the number of inserts in the data area may be changed.

NONE
All inserts defined previously are ignored.
This operand value may be specified only in conjunction with MF=M.

LAN='language'
Specifies the language to be used for message output. This operand is ignored if DEST=CONSOLE is specified.

'language'
1 letter to identify the language: D = German, E = English.
For additional options please consult system administration. The default value is defined by the system parameter MSGLPRI (see the “Introduction to System Administration“ [10]); this value is also assumed in the event of invalid specifications.

REPLY=
Defines a reply area. The area must be aligned on a halfword boundary and begin with a record length field (4 bytes: bytes 1-2: length of the reply; bytes 3-4 reserved). Before the macro is executed, bytes 1-2 must contain the length of the reply area (≤ 4095 bytes). When the macro is executed, the current length of the reply is entered in bytes 1-2.

Lowercase letters are converted to uppercase when entered via the REPLY operand. REPLY may be specified only in conjunction with DEST=SYSOUT/CONSOLE.

If “?” is entered as a reply and is reserved as a codeword in MIP, MIP displays the meaning of the message in question and the action required, before the message is output again in response.

(replylength,...)

`replylength`	length of the reply area > 4 bytes
`addr`	symbolic address (name) of the area
`(r)`	register containing the address of the area (may be specified only if MF=M)

((r1),(r2))

`r1` register containing the length of the reply area

`r2` register containing the address of the area

May only be specified if MF=M.

N

No reply area is generated in the CSECT/DSECT.

This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with MF=C/D.

DEST=

Designates destinations for the converted system message. Only DEST=SYSOUT / CONSOLE may be specified in conjunction with REPLY.

SYSOUT

Output to SYSOUT. This is the default setting if another output location is not specified with BUFFER. If an output location is specified with BUFFER and output to SYSOUT is desired, DEST=SYSOUT must be specified explicitly.

If an S variable was declared and the variable stream was directed into it, and the message is a guaranteed one, output is also to the S variable.

SYSLST

Output to SYSLST.

CONSOLE

Output to the operator console.

(destination,...)

Combination of the above-mentioned destinations; entry in parentheses, separated by commas.

NONE

Deletes the destinations SYSOUT, SYSLST and CONSOLE from the operand list. This operand value may be specified only in conjunction with MF=M. Note that the operand list must always contain at least one destination, e.g. a special area designated for this purpose (see the BUFFER operand).

BUFFER=

Specifies an area to which the converted system message is to be transferred. If output to SYSOUT is also desired, DEST=SYSOUT must be explicitly specified.

BUFFER must be aligned on a halfword boundary. A record length field is entered in the first 2 bytes (WROUT-Format):

`Byte 0-1`	length of the area
`Byte 2-3`	reserved
`Byte 4`	output control character
`Byte 5-n`	message text

(bufferlength,...)
bufferlength = length of the area > 16 bytes
`addr` symbolic address (name) of the area
`(r)` register containing the address of the area (may be specified only if MF=M)
The sum of the area lengths must be ≤ 4095 bytes.

((r1),(r2))
`r1` register containing the length of the message area.
`r2` register containing the address of the area.
May only be specified if MF=M.

N
No area for the converted system message is generated in the CSECT/DSECT. This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with MF=C/D.

MAP=
Specifies whether BUFFER is given another structure (mapping format).

NO
BUFFER is created in the **WROUT** format.

YES
BUFFER is created in the mapping format (see below).

UCDEST=
UCON destination (UCON=**U**niversal **Con**sole); (see the “Introduction to System Administration” [10]).

The destination of a message can be specified as follows:

- mnemonic device name for a particular console
- routing code for consoles and authorized user tasks which are allocated a particular area of activity
- authorization name for an authorized user task.

ℹ UCDEST is not executed unless DEST=CONSOLE was specified.
UCDEST has priority over the routing code attribute of the message in the message file.

'destcode'
The following entries are permissible for 'destcode':

- '(mn)'
  mn: 2-character mnemonic device name.
- '< x'
  x: routing code; The < character must be specified.
- 'name'
  name: name of the user task (4 characters).

destaddr
Symbolic address (name) of an area (4 bytes) with the entry for “destcode”. Left-justified entries; alignment on a word boundary.

(r)
Register containing the address of the area (4 bytes). May be specified only if MF=M.

N
No field for “destcode” is to be generated in the CSECT/DSECT.
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with MF=C/D.

DMS=
Specifies the message search mechanism.

APPL
The message is sought with the aid of DMS (message files and DLAM area).

NOTAPPL / NA
The message is sought only at system level in the DLAM area (without DMS). Reply if the requested message is not found:
`msgid,(DMS NOT IN MEMORY),<program pointer>,<modul>,<inserts>`

BUFFUSE=
Specifies whether the message is to be stored in an S variable via a variable stream (SYSMSG) and can be output by means of the HELP-MSG-INFORMATION MSG-ID= *LAST command (in other words, without explicitly specifying the message number).
Specifying BUFFUSE is relevant only of the output does not go to SYSOUT (DEST ≠ SYSOUT) and a transfer area was specified with the BUFFER operand.
If the message is (also) output to SYSOUT, and if it is a guaranteed message, it is always output to the S variable (where one was declared and the variable stream was directed into it). The variable stream includes only “guaranteed messages”. See the manuals “SDF-P” [21] and “Utility Routines (MSGMAKER)” [27].

INTERNAL
Output is not directed to a variable stream. The message cannot be output by means of the HELP-MSG-INFORMATION MSG-ID=*LAST command (in other words, without explicitly specifying the message number.

EXTERNAL
If an S variable was declared and the variable stream directed to it (with the command ASSIGN-STREAM or EXECUTE-COMMAND or with the CMD or OPSGEN macro), the message is output to this S variable.
The message code, the text of the message and the default values of the inserts are output. The message must be a guaranteed one, since otherwise it is not included in the variable stream. If the message is the last one output by the task, it can be output with the command HELP-MSG-INFORMATION MSG-ID=*LAST.

DEFTEXT=

Refers to a default message text. This default message text is output if the required message is not defined or if the required message or MIP task is not currently available. A corresponding return code is output. The default message text must be ≤ 4079 bytes.

NONE

No default message text specified.

(textlength,...)

`textlength`	= length of area
`addr`	symbolic address (name) of area
`(r)`	register containing address of area (may be specified only if MF=M)

((r1),(r2))

`r1`	register containing length of default message text.
`r2`	register containing address of area.

May be specified only if MF=M.

TIMER=

Specifies the maximum waiting time for a reply (REPLY operand) after a message is sent to SYSOUT.
If no reply is received within the time specified, the corresponding return code is output.

UNLIMITED

Unlimited waiting time.

(r)

r: register containing the waiting time. May be specified only if MF=M.

value

Waiting time, which can be between 10 and 3600 seconds.

TIMESTAMP=

Allows additional output of local system time.

NO

Local system time is not output.

YES

Local system time is output in ISO4 format in addition to the message. The output then has the following format:

```
%␣␣hh:mm:ss msgid text
```

where:

`hh`	hour
`mm`	minute
`ss`	second
`msgid`	message ID
`text`	message text

Note

For representation of time stamps, see also **CTIME** macro (page 356).

If MIP cannot read a time due to an internal error, the following is output:

```
%␣␣HH:MM:SS msgid text
```

DATESTAMP=

Allows additional output of date.

NO

Date is not output.

YES

The date is output in ISO4 format in addition to the message. The output then has the following format:

```
%␣␣yyyy-mm-dd msgid text
```

where:

`yyyy`	year
`mm`	month
`dd`	day of the month
`msgid`	message ID
`text`	message text

Note

For representation of time stamps, see also **CTIME** macro (page 356).

If MIP cannot read a date due to an internal error, the following is output:

```
%␣␣YYYY-MM-DD msgid text
```

If both date and time output is required, the date comes before the local system time.

MF=

For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

Notes

- The total length of a message including the message code must be ≤ 4079 bytes. Any characters in excess of this are automatically cut.
- If output is to the console, the maximum length of a message, including message key is only 230 bytes.
- Trailing blanks are automatically suppressed.
- Message output to SYSOUT, SYSLST or to a BUFFER area always begins with the character `%` (e.g. `%␣␣NMH1121 <text>`).
  If, however, the message is output to the console, it begins with a blank before the `%` (e.g. `␣%␣␣NMH1121 <text>`).

Mapping format

The mapping format specifies the structure of the BUFFER area in the case of MAP=YES. Various entries (mapping list) are prefixed to the structure defined under BUFFER.

Structure of the mapping format:

`Bytes 0 - 1`	length of the mapping list
`Bytes 2 - 3`	C'MP'
`Bytes 4 - n`	entries
`Bytes n+1 - p`	remainder of structure as under BUFFER (WROUT format).

The entries contain information on

- the inserts
- the message code
- the routing code
- the message weight
- as necessary, slack bytes for alignment of the WROUT buffer.

The entries are stored in the formats below:

<table>
<tr><th>Entry on</th><th>Structure of the entry</th></tr>
<tr><td>Insert</td><td><table><tr><td>Insert length</td><td>Displacement from insert in WROUT format</td><td></td></tr><tr><td>0H</td><td>2</td><td>4</td></tr></table>(A separate entry ... is created for each entry.)</td></tr>
<tr><td>Message code</td><td><table><tr><td>X' 81'</td><td>7-character message code</td><td></td></tr><tr><td>0H</td><td>1</td><td>8</td></tr></table></td></tr>
<tr><td>Message weight</td><td><table><tr><td>X' 20'</td><td>Message weight</td><td></td></tr><tr><td>0</td><td>1</td><td>1</td></tr></table></td></tr>
<tr><td>Routing code</td><td><table><tr><td>X' 50'</td><td>Routing code (left-justified)</td><td></td></tr><tr><td>0</td><td>1</td><td>5</td></tr></table></td></tr>
<tr><td>Slack bytes</td><td><table><tr><td>X' 00'</td><td>- - -</td><td>X' 00'</td><td>WROUT format</td></tr><tr><td>0</td><td>1</td><td>m</td><td>H</td></tr></table></td></tr>
</table>

Notes on the macro call

- If output is in mapping format, an entry with length =0 and address=0 stands for an insert which does not exist. Such entries are output only if the inserts of a message are not consecutively numbered.
- Inserts for which no current value is available, however, are described by an entry with length=0 and the distance to the defined insert position. The default text of the insert or, if no default text is defined, the standard insert (&xx) is found starting at this distance.
- If another MSG7X parameter list for an input is to be constructed from the mapping output, the distance for non-specified inserts must be reset to 0 because all other distance values in conjunction with length=0 are regarded as defining an insert which begins with a 4-byte record length field.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the MSG7X macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode; Subcode2 is only used with an I/O error and then contains the main return code of the **WROUT** or **WRTRD** macro):

X'cc'	X'bb'	X'aaaa'	Meaning
	X'00'	X'0000'	MSG7X macro executed successfully
X'cc'	X'04'	X'0001'	Abnormal termination of output (SYSOUT, SYSLST or CONSOLE). In this case Subcode2 (X' cc') contains the Maincode of the WROUT or WRTRD macro
	X'08'	X'0001'	Operand error: invalid specification for message number, area address, mnemonic device name, name of the authorized user task or area length
	X'0C'	X'0001'	Invalid request for a reply, e.g. in batch mode or when nonspecifiable destinations (SYSLST or several destinations) are designated in the DEST operand
	X'10'	X'0001'	No memory space available
	X'14'	X'0001'	BREAK during execution of the WROUT macro
	X'18'	X'0001'	Message text was truncated on transfer to the output area
	X'20'	X'0001'	Message output was aborted
X'00'	X'41'	X'FFFF'	MIP subsystem not loaded
X'02'	X'00'		Message not defined
X'03'	X'00'		MIP task not available
X'05'	X'00'		DMS subsystem not loaded
X'06'	X'00'		Message not available. Error in file

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

The calling program is terminated when the following errors occur:

- The data area is not assigned to the caller.
- The data area is not aligned on a word boundary.
- The data area is protected against write access.

Example 1

This example shows various options for seeking and outputting messages.

```
MSG7X1   START
MSG7X1   AMODE ANY
MSG7X1   RMODE ANY
         PRINT NOGEN
         BALR  3,0
         USING *,3
*
         SYSFL 'SYSLST=LST.MSG7X'
MFE1     MSG7X MF=E,PARAM=MFL1 ──────────────────────────────────────  (1)
MFE2     MSG7X MF=E,PARAM=MFL2 ──────────────────────────────────────  (2)
         WROUT TEXT,ERROR,PARMOD=31
MFE3     MSG7X MF=E,PARAM=MFL3 ──────────────────────────────────────  (3)
         WROUT TEXT,ERROR,PARMOD=31 ─────────────────────────────────  (4)
         SYSFL 'SYSLST=(PRIMARY)'
*
ERROR    TERM
*
MFL1     MSG7X MF=L,ID=DMS0E27,DMS=NOTAPPL,LAN='D'
MFL2     MSG7X MF=L,ID=DMS0E27,DEST=(SYSLST,SYSOUT),LAN='D'
MFL3     MSG7X MF=L,ID=SCP0976,DEST=SYSLST,BUFFER=(75,TEXT)
*
TEXT     DC    Y(TEXTEND-TEXT)
         DS    3X
         DS    0CL75
         DC    C'message output via WROUT: '
TEXTEND  EQU   *
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,msg7x1), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,msg7x1))
%  ASS6011 ASSEMBLY TIME: 508 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 82 MSEC
//end
%  ASS6012 END OF ASSEMBH
```

```
/start-executable-program library=macexmp.lib,element-or-symbol=msg7x1
%  BLS0523 ELEMENT 'MSG7X1', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'MSG7X1', VERSION ' ' OF '<date> <time>' LOADED
%  DMS0E27   ( DVS NICHT VERFUEGBAR )  --PC=00000034 IN
   P1-MODULE=********-------------------------------------------------  (5)
%  DMS0E27 FEHLER BEIM SCHLIESSEN EINER DATEI. EIN- AUSGABE MIT
   HARDWARE-FEHLER BEENDET -------------------------------------------  (6)
message output via WROUT:
%  SCP0976 LOGICAL VALIDATION PROBLEM DURING COMMAND PROCESSING ------  (7)
```

Output on SYSLST:

```
%  DMS0E27 FEHLER BEIM SCHLIESSEN EINER DATEI. EIN- AUSGABE MIT
   HARDWARE-FEHLER BEENDET
%  SCP0976 LOGICAL VALIDATION PROBLEM DURING COMMAND PROCESSING
```

(1) The message DMS0E27 should be output to SYSOUT. The message should only be sought in the DLAM area (without DMS). It is not found in the DLAM area, so the message is not output; A corresponding return code is output.

(2) The message DMS0E27 should be output to SYSOUT and to SYSLST. The restriction under (1) does not apply.

(3) The message SCP0976 should be output to SYSLST and written to the specified area TEXT.

(4) The macro **WROUT** writes the contents of the macro TEXT to SYSOUT.

(5) Output of the message from (1). The message is output in German because of the LAN=’D’ operand.

(6) Output of the message from (2). The message is output in German because of the LAN=’D’ operand.

(7) Output of WROUT.

Example 2

The following example shows message output being directed to an S variable. The S variable must be declared before calling the program and the variable stream SYSMSG must be assigned to it. Only the guaranteed messages are included in the SYSMSG variable stream.

```
MSG7X2   START
MSG7X2   AMODE ANY
MSG7X2   RMODE ANY
         PRINT NOGEN
         BALR  3,0
         USING *,3
*
MFE1     MSG7X MF=E,PARAM=MFL1
MFE2     MSG7X MF=E,PARAM=MFL2
MFE3     MSG7X MF=E,PARAM=MFL3
MFE4     MSG7X MF=E,PARAM=MFL4
*
ERROR    TERM
*
****  Definitions  ****
*
MFL1     MSG7X MF=L,ID=CMD0500,BUFFER=(250,BUF),MAP=YES,            *
               DEST=(SYSOUT,SYSLST) ——————————————————————————————  (1)
MFL2     MSG7X MF=L,ID=DMS0DF8,BUFFER=(250,BUF),MAP=YES,DEST=SYSOUT —  (2)
MFL3     MSG7X MF=L,ID=SCP0976,BUFFER=(250,BUF),DEST=SYSLST —————————  (3)
MFL4     MSG7X MF=L,ID=DMS0574,BUFFER=(250,BUF),DEST=SYSLST,        *
               BUFFUSE=EXTERNAL ——————————————————————————————————  (4)
*
BUF      DS    0CL250
         DC    Y(ENDBUF-BUF)
         DS    3X
         DS    CL245
ENDBUF   EQU   *
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,msg7x2), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,msg7x2)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 405 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 80 MSEC
//end
%  ASS6012 END OF ASSEMBH
/declare-var msg(type=structure),mult-elem=*list ───────────────────────  (5)
/assign-stream sysmsg,to=*var(msg)
/start-executable-program library=macexmp.lib,element-or-symbol=msg7x2
%  BLS0523 ELEMENT 'MSG7X2', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'MSG7X2', VERSION ' ' OF '<date> <time>' LOADED
%  CMD0500 INVALID DESCRIPTION OF COMMAND OR STATEMENT IN CURRENT SYNTAX FILE
%  DMS0DF8 EXPECTED VSN '(&01)' FOR FILE '(&02)', VSEQ '(&03)' NOT MOUNTED ON
 DEVICE '(&00)'. VSN '(&04)' FOUND INSTEAD. REPLY (0=EXIT; 1=RETRY; 2=DISPLAY
 LABEL;  =ACCEPT)
/assign-stream sysmsg,to=*dummy ─────────────────────────────────────────  (6)
/show-var msg
MSG(*LIST).MSG-TEXT = %  BLS0523 ELEMENT 'MSG7X2', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
MSG(*LIST).MSG-ID = BLS0523
MSG(*LIST).I0 = MSG7X2
MSG(*LIST).I1 = @
MSG(*LIST).I2 = :2OSG:$QM212.MACEXMP.LIB
MSG(*LIST).MSG-TEXT = %  BLS0524 LLM 'MSG7X2', VERSION ' ' OF
   '<date> <time>' LOADED
MSG(*LIST).MSG-ID = BLS0524
MSG(*LIST).I0 = MSG7X2
MSG(*LIST).I1 =
MSG(*LIST).I2 = <date> <time>
MSG(*LIST).MSG-TEXT = %  BLS0551 COPYRIGHT (C) . . .
MSG(*LIST).MSG-ID = BLS0551
MSG(*LIST).I0 = FUJITSU TECHNOLOGY SOLUTIONS
MSG(*LIST).I1 = 2012
MSG(*LIST).MSG-TEXT = %  CMD0500 INVALID DESCRIPTION OF COMMAND OR STATEMENT
   IN CURRENT SYNTAX FILE
MSG(*LIST).MSG-ID = CMD0500
MSG(*LIST).MSG-TEXT = %  DMS0574 DMS ERROR CODE '(&00)' OCCURRED WHEN
```

```
   DELETING SYSTEM FILE. COMMAND NOT PROCESSED
MSG(*LIST).MSG-ID = DMS0574
MSG(*LIST).IO = (&00)
```

(1) The message with message code CMD0500 is output to SYSOUT and SYSLST and to the BUF area, from where it is written to the S variable ABC via the variable stream SYSMSG. The output area BUF is structured in the mapping format.

(2) The message with message code DMS0DF8 is output to SYSOUT and the BUF area in mapping format. However, since it is not a guaranteed message, it is not included in the variable stream SYSMSG, and so is not written to ABC.

(3) The message with message code CMD0800 is output to SYSLST and the BUF area. BUFFUSE is set to INTERNAL as the default; it is not included in the variable stream SYSMSG or the S variable ABC.

(4) The message with message code DMS0574 is output to SYSLST and the BUF area, from where it is written to the S variable ABC via the variable stream SYSMSG.

(5) A composed “List”-type S variable called ABC is declared. Each guaranteed message is stored in the structured S variable ABC as a list element. Elements are added to the S variable until the assignment to SYSMSG is terminated.

(6) The assignment of the variable stream SYSMSG to the S variables ABC is canceled. The contents of the S variables ABC are displayed.

In addition to the messages written by the program in the S variable ABC, the S variable also contains guaranteed messages which were output when loading the MSG7X2 module. These were also output to the S variable ABC when loading the MSG7X2 module after the redirection of the variable stream SYSMSG.

MSGRC – Output return codes

General

Application area: Messages; see page 161
Macro type: Definition macro; see page 28

Macro description

The **MSGRC** macro outputs the return codes and their explanations in (equate) list form for the following macros described in this manual:

MSG7 MSGSINIT MSGSHOW MSGSMOD

The macros can be specified individually or in a list form. **MSGRC** does not necessarily output the return codes in the same order.

Macro format and description of operands

MSGRC
P=I / p ,FUNCT=ALL / macro / (macro, ...,macro)

P=
Prefix to all the symbolic names in the list.

I
All symbolic names begin with `I`.

p
A single letter that is to be used as a prefix.

FUNCT=
Specifies the macros whose return codes are to be listed.

ALL
All macros of the “messages” function group are listed.

macro
Name of the macro whose return codes are to be output.

(macro, ..., macro)
List of macro names whose return codes are to be output.

List for the macros MSG7, MSGSHOW, MSGSINIT and MSGSMOD

```
           MSGRC  P=A,FUNCT=(MSGSMOD,MSG7,MSGSHOW,MSGSINIT)
1          #INTF INTCOMP=1,INTNAME=MIP-MSG7,REFTYPE=REQUEST
1 ********        ********
1 ********  MSG7  ********
1 ********        ********
1 *
1 AM7OK      EQU   X'00'      MSG7 PROCESSED SUCCESSFULLY
1 AM7IOERR   EQU   X'04'      I/O ERROR
1 AM7MBINC   EQU   X'08'      PARAMETER LIST ERROR
1 AM7REPBA   EQU   X'0C'      REPLY REQUIRED IN BATCH PROCESSING
1 AM7RQMER   EQU   X'10'      NO MEMORY AVAILABLE TO PROCESS THE FUNCTION
1 AM7BRKWR   EQU   X'14'      BREAK DURING THE WROUT MACRO
1 AM7TRUNC   EQU   X'18'      MESSAGE-TEXT TRUNCATED
1 AM7MOINT   EQU   X'20'      MESSAGE OUTPUT PROCESSING INTERRUPTED
1 AM7RCINC   EQU   X'24'      INCORRECT ROUTING-CODE
1 AM7RPLST   EQU   X'2C'      REPLY INCORRECT WHEN SYSLST REQUIRED
1 AM7RPMD    EQU   X'30'      REPLY INCORRECT WHEN MANY OUTPUT
1 *                           DESTINATIONS GIVEN
1          SPACE 3
1 ********           ********
1 ******** MSGSINIT  ********
1 ********           ********
1 *
1 ASIOK      EQU   X'00'      MSGSINIT PROCESSED SUCCESSFULLY
1 ASIUNRES   EQU   X'04'      FILE: NO FILE AVAILA&BLE.
1 *                         TRACE: REQUIRED STATUS ALREADY EXISTING
1 ASIPLERR   EQU   X'08'      PARAMETER LIST ERROR
1 ASINTSOS   EQU   X'0C'      USER NOT TSOS
1 ASINOTPR   EQU   X'10'      SYSTEM UNABLE TO PROCESS THE MACRO
1          SPACE 3
1 ********             ********
1 ********   MSGSMOD   ********
1 ********             ********
1 *
1 ASMOK      EQU   X'00'         MSGSMOD PROCESSED SUCCESSFULLY
1 ASMERRDP   EQU   X'04'         MSGSMOD ERROR DURING PROCESS
1 ASMPLERR   EQU   X'08'         MSGSMOD PARAMETER LIST ERROR
1 ASMNTSOS   EQU   X'0C'         MSGSMOD USER NOT TSOS
1 ASMRQMER   EQU   X'10'         NO MEMORY AVAILABLE ($REQM ERROR)
1 ASMNOFIL   EQU   X'24'         MSGSMOD NO MESSAGE FILE
1 ASMNOTPR   EQU   X'28'         MSGSMOD UNABLE TO PROCESS
1          SPACE 3
```

```
1 ********            ********
1 ********   MSGSHOW   ********
1 ********            ********
1 *
1 ASHOK      EQU   X'00'         MSGSHOW PROCESSED SUCCESSFULLY
1 ASHERRDP   EQU   X'04'         ERROR DURING PROCESS
1 ASHPLERR   EQU   X'08'         MSGSHOW PARAMETER LIST ERROR
1 ASHNTSOS   EQU   X'0C'         BUFFER TOO SHORT
1 ASHRQMER   EQU   X'10'         NO MEMORY AVAILABLE ($REQM ERROR)
1 ASHUNBLE   EQU   X'30'         UNABLE TO PROCESS (INTERNAL ERROR)
```

MSGSHOW – Output information about system- or task-specific message files

General

Application area:	Messages; see page 161
Macro type:	Type S, MF format **1**: standard/L/E/C/D form; see page 29

Message files can be assigned a scope (system-wide or task-specific). Nonprivileged users can use their own message files for message output restricted to their own tasks. In addition, it is possible to specify the language which is to be given preference when selecting the message texts for output. Message files and the language specification are included in the message system either by means of the **MSGSMOD** macro or the MODIFY-MSG-FILE-ASSIGNMENT command.

The SHOW-MSG-FILE-ASSIGNMENT command corresponds to the **MSGSHOW** macro. For further information on the commands, see the “Commands” manual [19].

Macro description

The **MSGSHOW** macro provides information about the following:

- number of message files (system-wide, task-specific)
- language used for message output (system-wide, task-specific)
- names of message files; each name is preceded by an indicator of the access method (DLAM, ISAM). System message files are listed first, followed by the task-specific message files.

An example of the layout of the output is given following the description of operands.

Macro format and description of operands

MSGSHOW
BUFFER= (length,$\left\{\begin{array}{l}\text{addr}\\ \text{(r)}\end{array}\right\}$) ,SCOPE=BOTH / SYSTEM / TASK ,MF=S / C / (C,pre) / (E,...) / (D,pre) / D / L

BUFFER=
Specifies the length and address of an area for the output data. The area must be aligned on a word boundary. The **MSGDSHL** macro serves to generate a description (DSECT/data list) of this output area.

length
Length of the area in bytes; “length” ≥ 16. If the area is too small, only the number of message files is entered; see “Return information and error flags”, below, RC = X'0C'.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

SCOPE=
Specifies whether system-wide or task-specific message files are to be listed.

BOTH
Both system and task-specific message files are listed.

SYSTEM
Only the system message files are listed.

TASK
Only the task-specific message files are listed.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
In the C form or D form, a prefix (pre = 1..4 letters) can be specified, as shown in the macro format.

Default values:	pre = C for C form
	pre = D for D form

If less than four letters are specified for a prefix, the string SHOx results (where x ≙ first letter).

Layout of the output

MSGDSHL
[C/D][,p]

The **MSGDSHL** macro serves to generate either a data list of the output area or a dummy section (DSECT) of the output area.

C/D
A data list/DSECT is generated.

p
Prefix for the symbolic names of the DSECT/data list. No more than the first 3 characters of the string specified here will actually be used as a prefix.
Default value: p=SHL.

Register contents

Register R1 contains the operand list address.
Register R15 contains the return code.

Return information and error flags

R15:	0	0	0	0	0	0	a	a

A return code relating to the execution of the MSGSHOW macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Normal execution
X'04'	Error during macro execution
X'08'	Operand error
X'0C'	Output area too small
X'30'	Macro cannot be executed
X'41'	MIP subsystem is not loaded

DSECT of the output area

```
          MSGDSHL D
1         #INTF INTCOMP=1,INTNAME=MIP-SHOW,REFTYPE=REQUEST
1 SHLD     MFPRE MF=D,PREFIX=*NONE,DNAME=SHLD,MACID=SHL,DMACID=SHL,      C
1                ALIGN=D
2 SHLD     DSECT ,
2                *,##### PREFIX=, MACID=SHL #####
1 SHLDBUFL DC    Y(SHLDMLEN)           --LENGTH OF THE BUFFER
1 SHLDSYSN DC    X'00'                 --SYSTEM FILE NUMBER
1 SHLDTSKN DC    X'00'                 --TASK FILE NUMBER
1 SHLDSYSL DC    CL3' '                --SYSTEM LANGUAGE
1 SHLDTSKL DC    CL1' '                --TASK LANGUAGE
1 SHLDSEAR DC    X'00'                 --SEARCH VALUE
1 SHLDSALL EQU   X'80'                 --SEARCH = *ALL
1 SHLDSTSK EQU   X'20'                 --SEARCH = *TASK
1 SHLDRES  DS    CL7                   --RESERVED
1 SHLDFIXD EQU   *-SHLD                --FIXED-PART LENGTH
1 SHLDFLST DS    510CL55               --FILE NAMES LIST
1          ORG   SHLDFLST
1 SHLDFNAM DS    CL55                  --1ST FILE
1          ORG   SHLDFNAM
1 SHLDINDT DS    CL1                   --FILE TYPE
1 SHLDTIDL EQU   X'80'                 --DLAM + ISAM
1 SHLDTISA EQU   X'40'                 --ISAM
1 SHLDTDLA EQU   X'20'                 --DLAM
1 SHLDTLDL EQU   B'00010000'           --LOCAL DLAM + ...
1 SHLDNAME DS    CL54                  --FILE NAME
1          ORG
1 SHLDMLEN EQU   *-SHLD                --MAX LENGTH OF THE BUFFER
```

MSGSINIT – Lock message file or add message file to message system

General

Application area:	Messages (system administration macro); see page 161
Macro type:	Type S, MF format **1**: 31-bit interface: standard/L/E/C/D form; see page 29

System messages are assigned to the message files that contain them by means of the global class list. The class list contains all message classes (the first 3 characters of the message code) and the assigned names of the message files. The class list is created at system start time, and can be modified by system administration during normal operation using the **MSGSINIT** / **MSGSMOD** macros, or the MODIFY-MSG-FILE-ASSIGNMENT command.
A message file consists of a message work file and the corresponding HELP file (reduced message primary file).

Macro description

The **MSGSINIT** macro enables system administration to add a further message file to the message system or to prohibit access to a message file. The message class and name of the new message file are entered at the beginning of the class list. All references to a message file which is to be locked are deleted from the class list.
Modification of this list applies only to the current system run; the generation values are not changed. **MSGSINIT** enables system administration to activate or deactivate the MIP trace function.

Macro format and description of operands

MSGSINIT
[FILE=ADD / DEL / STD] [,TRACE=ON / OFF] ,MF=S / C / (C,pre) / (E,...) / (D,pre) / D / L

FILE=

This operand refers to the message file that is to be added or locked. The link name SMSGFILE must be assigned to the appropriate message work file before the macro call.

ADD
Message classes and the name of the message file are added at the beginning of the global class list.

DEL
All references to the message file to be locked are deleted from the class list.

STD
The STARTUP state is created for the class list.

TRACE=
The operand switches the MIP trace function on or off.

ON
The trace function is activated.

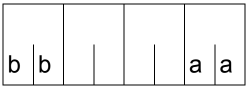
OFF
The trace function is deactivated.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A prefix (pre = 1 letter) can be specified in the C form or D form, as shown in the macro format. This prefix is inserted into all symbolic names of the CSECT/DSECT as the 4th character (after “INI”).

Default values: pre = C for C form
pre = D for D form

Return information and error flags

R15:



A return code relating to the execution of the MSGSINIT macro is transferred in the rightmost byte of register R15.

Ret.Code	Meaning
X'00'	Normal execution
X'04'	Error in the FILE operand: message file is not available or link name is not assigned
X'08'	Operand error
X'0C'	Error during access authorization check: the caller is not system administration (does not have the TSOS ID)
X'10'	Resource limit: macro cannot be executed
X'41'	MIP subsystem is not loaded

MSGSMOD – Lock message files or add message files

General

Application area:	Messages; see page 161
Macro type:	Type S, MF format **1**: 31-bit interface: standard/L/E/C/D form; see page 29

System messages are assigned to the message files that contain them by means of the global class list. The class list contains all message classes (the first 3 characters of the message code) and the assigned names of the message files. The class list is created at system start time and can be modified by system administration during normal operation using the macros **MSGSINIT** and **MSGSMOD**, or the MODIFY-MSG-FILE-ASSIGNMENT command.
Nonprivileged users can use their own message files for message output, restricted to their own tasks. The names of the system-wide and/or task-specific message files can be requested by means of the **MSGSHOW** macro or the SHOW-MSG-FILE-ASSIGNMENT command. The default value for the language of the message output is defined in the user catalog, see the SHOW-USER-ATTRIBUTES command, DFAULT-MSG-LANGUAGE output field; if no default value is specified there, the value set in the startup parameter service is used. For further information on the commands, see the “Commands” manual [19].

Macro description

The **MSGSMOD** macro enables system administration to add further message files to the message system or to prohibit access to message files. The message classes and the names of the new message files are entered at the beginning of the class list (range assignment table). All references to message files which are to be locked are deleted from the class list. Modifications to this list are applicable only during the current system run - the generation values are not changed.
Nonprivileged users can add their own message files to the message system and define the language to be used for message output. These message files can then be used for the current task; they are accessed before the system message files when searching for messages.

One macro call can be used to add up to 8 message files to the message system (using the IMPORT operand), and to lock a maximum of 8 message files (using the EXPORT operand). If both EXPORT and IMPORT are included in the same call, the same form of parameter must be used for transmission of the file names in each operand (either register format (r), or addresses (addr), or file names (file)).

Macro format and description of operands

```
MSGSMOD

            { { (r)                     } }
            { { (r),(r), ... ,(r)       } }
            { { (REG,n)                 } }
            {                             }
            { { adr                     } }
[EXPORT=    { { (addr,addr, ... ,addr)  } }]
            { { (ADR,n)                 } }
            {                             }
            { { 'file'                  } }
            { { ('file', ... ,'file')   } }
            { { (FILE,n)                } }

            { { (r)                     } }
            { { (r),(r), ... ,(r)       } }
            { { (REG,n)                 } }
            {                             }
            { { addr                    } }
[,IMPORT=   { { (addr,addr, ... ,addr)  } }]
            { { (ADR,n)                 } }
            {                             }
            { { 'file'                  } }
            { { ('file', ... ,'file')   } }
            { { (FILE,n)                } }

[,SCOPE=TASK / SYSTEM]
,SEARCH=*UNCHANGED / *ALL / *STD / *TASK [1]
,LAN=*UNCHANGED / *STD / 'language'
,MF=S / C / (C,pre) / (E,...) / (D,pre) / D / L
```

[1] The SEARCH operand is no longer evaluated. It can still be specified for reasons of compatibility.

EXPORT=
IMPORT=
Specifies the message file(s) to be locked (EXPORT) or to be added to the message system (IMPORT).

(r)
Register containing the address value of a field which contains the name of a message file.

((r),..(r))
Specifies a list of up to 8 registers. Each register contains the address value of a field which contains the name of a message file.

(REG,n)
The registers to be used will be specified in the operand list. This operand can only be used if MF=L/D/C is specified.
n = number of addresses; n ≤ 8. Default value: n=1.

addr
Symbolic address of the field which contains the name of a message file.

(addr,...)
Specifies a list of up to 8 symbolic addresses. These addresses identify fields containing the names of the message files.

(ADR,n)
The addresses will be specified in the operand list. This operand can only be used if MF=L/D/C is specified.
n = number of addresses; n ≤ 8. Default value: n=1.

'file'
Name of a message file.

(file,...)
Specifies a list of up to 8 (message) file names.

(FILE,n)
The names of the message files will be specified in the operand list. This operand can only be used if MF=L/D/C is specified.
n = number of message files; n ≤ 8. Default value: n=1.

SCOPE=
Defines the scope of the message files specified with the IMPORT/EXPORT operands. The message files may be accessible to the entire system (i.e. all tasks in the system) or only to the task executing the calling program.

TASK
Only the task executing the program which issued the **MSGSMOD** macro can access the message files specified with IMPORT/EXPORT. The assignment is canceled automatically at the end of the task. EXPORT can be specified only for message file(s) previously assigned to the task by means of IMPORT.
TASK is the default value for nonprivileged users.

SYSTEM
All tasks in the system can access the message files specified with IMPORT/EXPORT. The assignment is valid for the current system run only. SYSTEM can be specified only under the system administration user ID.
SYSTEM is the default value for the system administration.

LAN=
Serves to define a language to be used for message output. This definition applies to the current task run only.

<u>*UNCHANGED</u>
The language defined for the task run is not changed.

***STD**
The language specified in the user catalog or with the system parameter MSGLPRI.

'language'
1 letter to identify the language, where D = German, E = English.
For symbols for other languages please consult system administration.

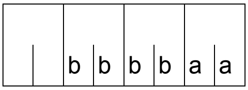
MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
The C form or D form can only be specified in conjunction with IMPORT/EXPORT=REG/ADR/FILE. A prefix (pre = 1..4 letters) can be specified, as shown in the macro format.

Default values: pre = MODC for C form
pre = MODD for D form

If less than four letters are specified for a prefix, the string MODx results (where x ≙ first letter).

Return information and error flags

R15:



A return code relating to the execution of the MSGSMOD macro is transferred in register R15. The values are hexadecimal constants.
bbbb must be read bit by bit.
Bit 2n-1 = 1: the n-th file (in the specified order, starting with EXPORT=...) could not be read without errors.

X'bbbb'	X'aa'	Meaning
X'xxxx'	X'00'	Normal execution
X'xxxx'	X'04'	Error during execution of the macro
	X'08'	Operand error
	X'0C'	Access authorization: user is not TSOS (system administration)
	X'10'	Resource limitation (REQM error)
X'xxxx'	X'30'	The macro cannot be executed
	X'41'	MIP subsystem is not loaded

NDGUINF – Output GS unit number

General

Application area:	Memory pools; see page 55
Macro type:	Type S, MF format **3**: C/D/L/M/E form; see page 29

Global storage (GS) is supplied with a maximum of two independent hardware units (GS units) which can be addressed from one or more servers.

GS is divided into so called partitions.

GS volumes are emulated volumes whose data is located exclusively in GS. They can be accessed from each system in the relevant XCS cluster. As far as the user is concerned, they function in much the same way as normal disk volumes.

High availability for a GS volume can be achieved if the volume is set up on a dual partition. For further information about GS, see the “Introduction to System Administration” [10].

Macro description

The **NDGUINF** macro indicates the GS unit on which the GS volume has been set up and whether the GS volume uses a dual partition. The information is transferred in the `NDGUGSU` and `NDGUFLG1` fields of the parameter list (see lower part of parameter list).

Macro formats and description of operands

NDGUINF
MF=C / D / L / M / E
,VSN=<var: char:6> / (<reg: A(char:6)>)
,PARAM=<var: pointer> / (reg: pointer>)
,PREFIX=N / p
,MACID=DGU / mac

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID, and PARAM), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see section “S-type macros” on page 29). No operands may be specified in the L form, otherwise a `MNOTE` will be output.

VSN=
Specifies the VSN (Volume Serial Number) of the volumes. Length: 1-6 characters. May only be specified in conjunction with MF=M.

<var: char:6>
Name of the field with the VSN.

(<reg: A(char:6)>)

Register with the address of the field containing the VSN.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the NDGUINF macro is transferred in the standard header (cc=subcode2,bb=subcode1,aaaa=main code)

X' cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally
X'00'	X'20'	X'0001'	An internal error occurred
X'00'	X'40'	X'0002'	The specified volume was not found
X'00'	X'01'	X'0003'	The parameter was not specified correctly
X'00'	X'40'	X'0004'	The VSN is ambiguous and the operator has rejected the volume

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Example

```
NDGUINF  START
         BALR  3,0
         USING *,3
         LA    4,PARLIST
         USING DSECT,4
*
         NDGUINF MF=M,VSN=VSNFIELD ------------------------------------  (1)
         NDGUINF MF=E,PARAM=PARLIST -----------------------------------  (2)
         TERM
*
****  DEFINITIONS  ****
VSNFIELD  DC    C'ABC123' ----------------------------------------------- (1)
PARLIST  NDGUINF MF=L --------------------------------------------------- (2)
DSECT    NDGUINF MF=D  --------------------------------------------------  (3)
         END
```

Extract from the parameter list

```
1 *
1 NDGUIN_DATA DS    0XL8                   Input parameters
1 NDGUVSN     DS    CL6                    Volume serial number
1 NDGURSV1    DS    XL2                    Reserved
1 *
1 *
1 NDGUOUT_DATA DS    0XL4                  Output parameters
1 NDGUFLG1    DS    AL1                    Indicator byte
1 NDGUDUAL    EQU   X'80'                  Set if volume in a dual
1 *                                        partition. In this case
1 *                                        gs_unit contains 0
1 NDGUNUSD    EQU   X'7F'                  Not used
1 NDGUGSU     DS    X                      GS-unit number
1 NDGURSV2    DS    XL2                    Reserved
1 *
1 NDGU#       EQU   *-NDGUHDR
```

Runtime log

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,ndguinf), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,ndguinf)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 396 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 94 MSEC
//end
%  ASS6012 END OF ASSEMBH
/load-executable-program library=macexmp.lib,element-or-symbol=ndguinf, -
/     test-options=*aid
%  BLS0523 ELEMENT 'NDGUINF', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'NDGUINF', VERSION ' ' OF '<date> <time>' LOADED
/%on %term<%d %4 ->ndgugsu;%d %4 ->ndguflg1;%d %4 ->ndguvsn>
/%r
*** TID: 005000D8 *** TSN: 2QSE ********************************************
SRC_REF:    31 SOURCE: NDGUINF  PROC: NDGUINF  *****************************
DSECT.NDGUGSU   = 00 ─────────────────────────────────────────────────  (4)
DSECT.NDGUFLG1  = 00   .
DSECT.NDGUVSN   = |ABC123|
```

(1) The **NDGUINF** macro is called in its M form. The associated data area is generated at the symbolic address VSNFIELD and the VSN is specified as the constant.

(2) The **NDGUINF** macro is called in its E form. The associated data area is generated at the symbolic address PARLIST by calling a **NDGUINF** macro with MF=L.

(3) A **NDGUINF** macro with MF=D generates the DSECT.

(4) The following information is transferred:

```
DSECT.NDGUGSU   = 00        : GS unit number
DSECT.NDGUFLG1  = 00   .    : Flag byte
DSECT.NDGUVSN   = |ABC123|  : Volume Serial Number (VSN)
```

NKDINF – Output data on (peripheral) configuration

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Type S, MF format **2**: standard/E/L/C/D/M form; see page 29

NDM (Nucleus Device Management) supplies information to the user, operator, and system administration on the allocation and availability status of the configuration and mounted data volumes. The configuration comprises the following: CPUs, channels, controllers, and devices.

Devices (disk devices, tape devices, printers, ...) are assigned to device types (T-C), device families (F-C), volume types (V-T) and device classes, see the “System Installation” [10] manual. A particular device can be addressed via its mnemonic device name (MN).

The **NKGTYPE** macro provides information about the names, device type codes, device characteristics, path addresses and path attributes for a particular device type or volume type, or about names and device type codes of the device types belonging to a device family or device class.

Macro description

The **NKDINF** macro enables access to data provided by the information services (NKD) of NDM. The data provides information on the following:

- the allocation level of a task
- the allocation and availability status of devices, device types device families, disks or tapes
- the structure of the configuration
- the device queue.

The **NKDINF** macro transfers the requested information in appropriately structured output records. The macro provides the user with the layouts of these output records as DSECTs to facilitate interpretation of the output.

The output records contain information on the allocation and availability status of the specified devices, hardware units or resources. The records are entered in an area of class 6 memory, which the macro has previously requested.

The output control record is also prefixed to the output records requested by the user in this output area; the output control record contains general information about macro execution and the structure of the output area. The start address of the area is passed in the `NKDIOPTR` field to the operand list.

The caller is responsible for returning the memory area (**RELM**). The length specifications should be taken from the output control record. The following should be noted when returning the memory area:

- the start address of the output buffer is aligned on a page boundary;
- the `NKDIOLEN` field in the output control record contains the size of the output area. The size is specified in main memory pages (4K) for nonprivileged callers.

The user can generate the layout of any output record offered by **NKDINF** as a DSECT using a macro call with MF=D and the RECORD operand. The DSECT permits symbolic addressing of the individual fields of an output record.

The user must change the evaluation of his CONFIG record when compiling existing programs (from a version < BS2000/OSD-BC V3.0) with BS2000/OSD-BC ≥ V3.0. The layout has changed to one which is incompatible with BS2000/OSD-BC V2.0, since the number of inner connections of the CONFIG been increased from 4 to 8.
In versions < BS2000/OSD-BC V3.0 compiled programs receive the output in the same format as before, and so are still able to run.

The following restrictions apply to nonprivileged users:

- task records are output only for tasks under the user's own ID,
- the caller is given no information on device queues (DVQ operand), allocation and reservation of specified device types (TYPTASK operand), disks for which explicit utilization specifications were made with the SET-DISK-PARAMETER command (DISC operand) and disks for which the DRV product is in use (DRV operand).
- Information on allocations by other users is masked out.
- The caller receives the output of the SUMMARY information only in the LOC record.

Macro format and description of operands

```
NKDINF

          ⎧ NO
          ⎪ ALL
          ⎪ CHN
          ⎪ CPU
          ⎪ CTL
          ⎪ DVC
          ⎪ IOSIDE
          ⎪ SE
          ⎪ SIDE
CONFIG=   ⎨   ⎧ ctl-mn    ⎫                                       ⎬
          ⎪   ⎪ chpid     ⎪
          ⎪   ⎪ chnrange  ⎪
          ⎪   ⎪ icuu      ⎪
          ⎪  (⎨ mn        ⎬,⎧ addr ⎫[,⎧ S ⎫])
          ⎪   ⎪ ioside#   ⎪ ⎩ (r)  ⎭  ⎩ L ⎭
          ⎪   ⎪ cpu#      ⎪
          ⎪   ⎪ se#       ⎪
          ⎪   ⎪ side#     ⎪
          ⎩   ⎩ dev#      ⎭

,EXTMN=NO / YES

        ⎧ NO
        ⎪ MONITORED
,DISC=  ⎨ SCHEDULED                                     ⎬
        ⎪ (⎧ mn  ⎫,⎧ addr ⎫[,⎧ S         ⎫])
        ⎩  ⎩ vsn ⎭ ⎩ (r)  ⎭  ⎩ L[,TASK]  ⎭

        ⎧ NO
        ⎪ ALL / (ALL,CAR)
,TAPE=  ⎨ (⎧ mn  ⎫,⎧ addr ⎫)                ⎬
        ⎪  ⎩ vsn ⎭ ⎩ (r)  ⎭
        ⎩ (mn,addr,CAR)

        ⎧ NO
,TASK=  ⎨ OWN                               ⎬
        ⎪ (⎧ tsn ⎫,⎧ addr ⎫)
        ⎩  ⎩ tid ⎭ ⎩ (r)  ⎭
```

NKDINF (cont.)

$$,\text{DEPOT}=\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{ALL}\\ (\left\{\begin{array}{l}\text{mn}\\ \text{location}\end{array}\right\},\left\{\begin{array}{l}\text{addr}\\ (\text{r})\end{array}\right\})\end{array}\right\}$$

$$,\text{DEVICE}=\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{ALL}\\ (\left\{\begin{array}{l}\text{mn}\\ \text{tt}\\ \text{fc}\end{array}\right\},\left\{\begin{array}{l}\text{addr}\\ \\ (\text{r})\end{array}\right\})\end{array}\right\}$$

,GLOBAL=$\underline{\text{NO}}$ / YES

$$,\text{UNMONIT}=\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{ALL}\\ (\text{vsn},\left\{\begin{array}{l}\text{addr}\\ (\text{r})\end{array}\right\})\end{array}\right\}$$

$$,\text{TYPTASK}=\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{ALL}\\ (\left\{\begin{array}{l}\text{fc}\\ \text{tt}\end{array}\right\},\left\{\begin{array}{l}\text{addr}\\ (\text{r})\end{array}\right\})\end{array}\right\}$$

$$,\text{SUMMARY}=\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{ALL}\\ (\left\{\begin{array}{l}\text{fc}\\ \text{tt}\end{array}\right\},\left\{\begin{array}{l}\text{addr}\\ (\text{r})\end{array}\right\})\end{array}\right\}$$

$$,\text{DRV}=\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{ALL}\\ \text{ALL-DRV}\\ (\text{vsn},\left\{\begin{array}{l}\text{addr}\\ (\text{r})\end{array}\right\})\end{array}\right\}$$

,DVQ=$\underline{\text{NO}}$ / YES

NKDINF (cont.)

$$\text{,LOC=}\left\{\begin{array}{l}\underline{\text{NO}}\\ \text{ALL}\\ (\left\{\begin{array}{l}\text{location}\\ \text{tt}\\ \text{fc}\\ \text{vt}\end{array}\right\},\left\{\begin{array}{l}\text{adr}\\ \\ \text{(r)}\end{array}\right\})\end{array}\right\}$$

[,RECORD=ALL / CONFIG / DEVICE / DEPOT / DISC / DRV / DVQ / GLOBAL / HEADER /
LOC / SUMMARY / TAPE / TASK / TYPTASK / UNMONIT]

,HWSTATE=NO / YES

,MF=S / E / L / C / D / M

[,PARAM=addr / (r)]

,PREFIX=N / p

,MACID=KDI / macid

The operands `CONFIG=IOSIDE/SE/SIDE/(ioside#,...)/(se#,...)/(side#,...)` return no information, since the associated hardware is no longer supported. The operand values can still be specified, for compatibilty.

The operands are described in alphabetical order below.

CONFIG=
One CONFIG output record is written for each specified unit of the configuration.

NO
This function is not desired.

ALL
Information relating to all generated hardware units is requested. A very large output area can occur with large configurations in the case of CONFIG=ALL.

CHN
Information relating to all channels is requested.

CPU
Information relating to all CPUs is requested.

CTL
Information relating to all multidevice controllers is requested.

DVC
Information relating to all devices is requested.

(ctl-mn,...)
The user transfers in a list (see page 673) the mnemonic names of the device controllers about which information is required.

(chpid,...)
The user transfers in a list (see page 673) the CHANNEL_PATH_ID of the channels about which information is required.

(chnrange,...)
The user transfers in a list (see page 673) the channel range about which information is required.

(icuu,...)
The user transfers in a list (see page 673) the device addresses of the hardware units about which information is required.
A device address designates the path on which a device can be addressed. The device address is 2 bytes long and comprises the following components:

Byte 1:	1st half-byte: number of the input/output processor 2nd half-byte: channel number
Byte 2:	port number of the multidevice controller and port number of the device, or only the port number of the device (if it is connected directly to the channel).

(mn,...)
The user transfers in a list (see below) the mnemonic device names of the hardware units on which information is required.

(cpu#,...)
The user transfers in a list (see below) the numbers of the CPUs on which information is required.

(dev#,...)
The user transfers in a list (see below) the device numbers of the hardware units on which information is required.

addr
Symbolic address (name) of a field. The field contains a list of mnemonic device names, device addresses or device numbers. It must be aligned on a word boundary.

(r)
Register containing the address value “addr”. A register may be specified only in conjunction with MF=M.

Format of the **list** for transferring mnemonic device names, device addresses or device numbers:

- The first word contains the number of entries in the list, right-justified in hexadecimal form.
- This is followed by the entries: each entry is 4 bytes long and contains (left-justified) the mnemonic device name, device address or device number. Bytes not required should be overwritten with X'00'.

S
Information is output in the short format (standard format).

L
Information is output in the long format and contains additional path descriptions.

DEPOT=
Provides information on the assignment of physical tape devices (mnemonics) to depots (locations).

NO
This function is not desired.

ALL
Information is output on all known locations.

(mn,...)
The user transfers in a list (see below) the mnemonic names of the tape devices on which information is required.

(location,...)
The user transfers in a list (see below) the locations on which information is required.

addr
Symbolic address of a field. The field contains a list of mnemonics or locations and must be aligned on a word boundary.

(r)
Register containing the address value “addr”. A register may be specified only in conjunction with MF=M.

Format of the **list** for transferring the (tape) mnemonics or locations:

- The first word of the list contains the number of entries in the list, right-justified in hexadecimal form.
- This is followed by the entries: For mnemonics, each entry is 4 bytes long. For locations, each entry is 8 bytes long. Each entry contains a mnemonic or location, left-justified.

DEVICE=

A DEVICE output record is output for each specified device.
See the note at RECORD=DEVICE, page 679.

<u>**NO**</u>

This function is not desired.

ALL

Information is requested on all devices.

(mn,...)

The user transfers in a list (see below) the mnemonic names of the devices about which information is required.

(tt,...)

The user transfers in a list (see below) the codes for the types of devices on which information is required.

(fc,...)

The user transfers in a list (see below) the codes for the families of devices about which information is required.

addr

Symbolic address (name) of a field. The field contains a list of mnemonic device names, codes for device families or device type codes. It must be aligned on a word boundary.

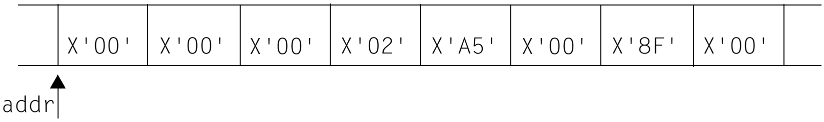
(r)

Register containing the address value “addr”. A register may be specified only in conjunction with MF=M.

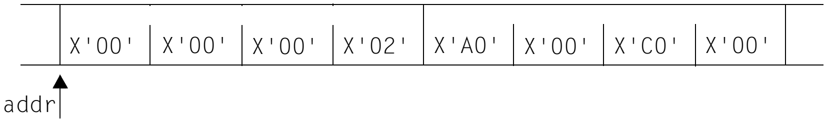
Format of the **list** for transferring family codes, device type codes or mnemonic device names:

- The first word contains the number of entries in the list, right-justified in hexadecimal form.
- This is followed by the entries: when family device type codes are specified each entry is 2 bytes long, when mnemonic device names are specified the length of the entry depends on the value of the EXTMN operand: with EXTMN=NO each entry is 2 bytes long, with EXTMN=YES each entry is 4 bytes long.
- The lists may contain a maximum 32 K of entries.

Example 1: 2 device types with device type codes X'A5' (= D3435) and X'8F' (= D3475)



Example 2: 2 device families with family codes X'A0' (= disk devices) and X'C0' (= MTC)



X'00'	X'00'	X'00'	X'02'	X'A0'	X'00'	X'C0'	X'00'

DISC=
Requests a DISC output record for each specified disk (disk device).

NO
This function is not desired.

MONITORED
Information is requested relating to all disks which are available online and monitored by NDM.

SCHEDULED
Information is requested relating to all disks for which explicit utilization specifications were made with the SET-DISK-PARAMETER command.
This value may be specified only under the system administration ID (TSOS).

(mn,...)
The user transfers in a list (see below) the mnemonic names of the disks about which information is required.

(vsn,...)
The user transfers in a list (see below) the VSNs (volume serial numbers) of the disks about which information is required.

addr
Symbolic address (name) of a field. The field contains a list of VSNs or mnemonic device names. It must be aligned on a word boundary.

(r)
Register containing the address value “addr”. A register may be specified only in conjunction with MF=M.

Format of the **list** for transferring mnemonic device names or VSNs:

- The first word contains the number of entries in the list, right-justified in hexadecimal form.
- This is followed by the entries: each entry is 8 bytes long and contains (left-justified) the VSN or the mnemonic device name. Bytes not required should be overwritten with X'00'.

S
Information is output in the short format (standard format).

L
Information is output in the long format and contains additional specifications about the SVL states and disk parameters.
This value may be specified only under the system administration ID (TSOS).

TASK
Output of a list of TSNs of the tasks that are currently working with the disk. The list consists of 4-byte entries and is output only for private disks in USE=DMS mode. TASK may be specified only in conjunction with the long format (L).
This value may be specified only under the system administration ID (TSOS).

DRV=
A DRV output record is requested for each disk specified for which the DRV product (Dual Recording by Volume; see the “DRV” manual [25]) is in use.

NO
This function is not desired.

ALL
A DRV output record is requested for each disk known to the DRV component.
This value may be specified only under the system administration ID (TSOS).

ALL-DRV
A DRV output record is requested for all disks for which the DRV operating mode is set.
This value may be specified only under the system administration ID (TSOS).

(vsn,...)
The user transfers a list (see below) of the VSNs (volume serial numbers) of the disks about which information is required.
This value may be specified only under the system administration ID (TSOS).

addr
Symbolic address (name) of a field. The field contains a list of VSNs. It must be aligned on a word boundary.

(r)
Register containing the address value “addr”. A register may be specified only in conjunction with MF=M.

Format of the **list** for transferring the VSNs:

- The first word contains the number of entries in the list, right-justified in hexadecimal form.
- This is followed by the entries: each entry is 8 bytes long and contains (left-justified) the VSN. Bytes not required should be overwritten with X'00'.

DVQ=
Specifies whether information on the device queue (secure queue) is to be output.

NO
This function is not desired.

YES
DVQ output records are written.
This value may be specified only under the system administration ID (TSOS).

EXTMN=
Specifies whether the caller transfers mnemonic device names in the old format (2 bytes long) or in the new format (4 bytes long) in the operand list, and/or expects mnemonic device names in the old or new format in the output area of the macro.
The macro provides fields in the output area for the 2-byte or 4-byte format to accept the mnemonic names. An indicator specifies for each output record whether the 2-byte or 4-byte output field has been filled.

NO
The user transfers only 2-byte mnemonic device names in the operand list and also only interprets 2-byte names.
If information is output for a device whose mnemonic name is 4 bytes long, the 2-byte field in the output area is deleted, the indicator for output in 4-byte format is set and the incompleteness of the information is indicated in the output prefix (OCR) and in the standard header through return codes.

YES
The user transfers and expects mnemonic device names always in 4-byte format.
If shorter mnemonic device names are specified, they are to be entered left-justified in the fields provided in the operand list and padded with blanks on the right.

GLOBAL=
Specifies whether the setting of all global NDMS control parameters is to be output.

NO
This function is not desired.

YES
An output record with the global NDM control parameters is written.

HWSTATE=
Outputs information about the hardware status (ON/OFF) of the unit.

ℹ Users of the CONFIG and CONFIG-L records who do not evaluate the fields `SIDE_/GP_` etc. `HARDWARE_STATE` (in NKDCUTYP of the CONFIG record) should call NKDINF with the HWSTATE=NO operand in order to speed up processing.

NO
This function is not desired.

YES
The hardware status of the unit is determined (only significant for special applications).

LOC=
Outputs the information scope of SUMMARY and TYPTASK, sorted according to location.

NO
This function is not desired.

ALL
An output record is created for each generated device type.

(location,...)
An output record is created for the specified locations for each generated device type. For the format of the location list, see DEPOT operand.

(tt,...)
The user transfers in a list (see below) the codes of the device types on which information is required.

(fc,...)
The user transfers in a list (see below) the codes of the device families on which information is required.

(vt,...)
The user transfers in a list (see below) the codes of the volume types on which information is required.

addr
Symbolic address of a field. The field contains a list of device type codes or family codes and must be aligned on a word boundary.

(r)
Register containing the address value “addr”. A register may be specified only in conjunction with MF=M.
Format of the **list** for transferring the device type codes or family codes:

– The first word of the list contains the number of entries in the list, right-justified in hexadecimal form.
– This is followed by the entries: each entry is 2 bytes long and contains (left-justified) the device type code or family code.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29.
The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

RECORD
May only be specified in conjunction with MF=C or MF=D and specifies for which output record a CSECT or DSECT is generated.

ALL
Generates CSECTs/DSECTs for all the output records (HEADER through DRV) offered by the macro.

CONFIG
Generate a CSECT/DSECT for the CONFIG output record (CONFIG operand).

DEPOT
Generates a CSECT/DSECT for the DEPOT output record (DEPOT operand).

DEVICE
Generates a CSECT/DSECT for the DEVICE output record (DEVICE operand).
Note
Given the functional extension “Dynamic I/O Configuration change”, the DEVICE record may now also contain records for those dummy devices which are present as placeholders in the BS2000 device table and which are subsequently replaced by real devices. The EXTENDED DEVICE MNEMONIC field in the records for these devices contains the mnemonic “DUMMY” and the DEVICE RECONFIGURATION STATE field contains the value X'0F' (INVALID).

DISC
Generates a CSECT/DSECT for the DISC output record (DISC operand).

DRV
Generates a CSECT/DSECT for the DRV output record (DRV operand).

DVQ
Generates a CSECT/DSECT for the DVQ output record (DVQ operand).

GLOBAL
Generates a CSECT/DSECT for the GLOBAL output record (GLOBAL operand).

HEADER
Generates a CSECT/DSECT for the output control record.
The output control record contains:

- pointers to the various output records
- counters
- length specifications (length of the output area, length of the individual output records) and
- return codes for the various output records.

The layout is reproduced at the end of the description.

LOC
Generates a CSECT/DSECT for the LOC output record (LOC operand).

SUMMARY
Generates a CSECT/DSECT for the SUMMARY output record (SUMMARY operand).

TAPE
Generates a CSECT/DSECT for the TAPE output record (TAPE operand).

TASK
Generates a CSECT/DSECT for the TASK output record (TASK operand).

TYPTASK
Generates a CSECT/DSECT for the TYPTASK output record (TYPTASK operand).

UNMONIT
Generates a CSECT/DSECT for the UNMONIT output record (UNMONIT operand).

SUMMARY=
A SUMMARY output record is requested for each specified device family (device type). The record contains summary information on the device family or the device type. For each device family, the set of SUMMARY records for its device types is supplied.

<u>NO</u>
This function is not desired.

ALL
A SUMMARY output record is requested for each device type defined in the system.

(fc,...)
The user transfers in a list (see below) the codes of the device families about whose device types information is required.

(tt,...)
The user transfers in a list (see below) the codes for the types of devices about which information is required.

addr
Symbolic address (name) of a field. The field contains a list of family codes or device type codes. It must be aligned on a word boundary.

(r)
r = register containing the address value “addr”. A register may be specified only in conjunction with MF=M.

Format of the **list** for transferring family codes or device type codes:

– The first word contains the number of entries in the list, right-justified in hexadecimal form.
– This is followed by the entries: each entry is 2 bytes long and contains (left-justified) the family code or device type code.

TAPE=
A TAPE output record is written for each specified tape device.

<u>NO</u>
This function is not desired.

ALL
Information is requested on all tapes monitored by NDM.

(mn,...)
The user transfers in a list (see below) the mnemonic names of the tape devices about which information is required.

(vsn,...)
The user transfers in a list (see below) the VSNs (volume serial numbers) of the tapes about which information is required.

addr
Symbolic address (name) of a field. The field contains a list of VSNs or mnemonic device names. It must be aligned on a word boundary.

(r)
r = register containing the address value “addr”. A register may be specified only in conjunction with MF=M.

Format of the **list** for transferring mnemonic device names or VSNs:

- The first word contains the number of entries in the list, right-justified in hexadecimal form.
- This is followed by the entries: each entry is 8 bytes long and contains (left-justified) the VSN or the mnemonic device name. Bytes not required should be overwritten with X'00'.

CAR
Additional records are output for 3591 Magnetic Tape Cartridges in “random” mode. These apply to cartridges already in the cartridge loader and which have already been registered by the device management system. Multiple records can be issued for a device.

TASK=
A TASK output record is written. The record contains information on devices, disks and tapes that are used by the specified task.

<u>NO</u>
This function is not desired.

OWN
Specifications refer to the calling task.

(tsn,...)
The user transfers in one word the TSN (task sequence number) of the task about whose device and volume allocation information is required.

(tid,...)
The user transfers in one word the TID (task identifier) of the task about whose device and volume allocation information is required.

addr
Symbolic address (name) of the word containing the TSN or TID of a task.

(r)
Register containing the address value “addr”. A register may be specified only in conjunction with MF=M.

TYPTASK=
One TYPTASK output record is requested for each specified device family (device type). The record indicates the number of devices of the specified type that have been reserved by a task. For each device family, the set of TYPTASK records relating to its device types is output.

<u>NO</u>
This function is not desired.

ALL
A TYPTASK output record is requested for all device types defined in the system.
This value may be specified only under the system administration ID (TSOS).

(fc,...)
The user transfers in a list (see below) the codes of the device families about whose device types information is required.
This value may be specified only under the system administration ID (TSOS).

(tt,...)
The user transfers in a list (see below) the device type codes of the device types about which information is required.
This value may be specified only under the system administration ID (TSOS).

addr
Symbolic address (name) of a field. The field contains a list of family codes or device type codes. It must be aligned on a word boundary.

(r)
Register containing the address value “addr”. A register may be specified only in conjunction with MF=M.

Format of the **list** for transferring family codes or device type codes:

- The first word contains the number of entries in the list, right-justified in hexadecimal form.
- This is followed by the entries: each entry is 2 bytes long and contains (left-justified) the family code or device type code.

UNMONIT=
An UNMONIT output record is requested for each of the tape volumes specified that has been reserved “offline”. This record indicates the VSN of the tape volume reserved as well as the TSN of the task which reserved it. “Offline” reservation means that a device/volume has been requested which is not (yet) monitored by NDM.

NO
This function is not desired.

ALL
An UNMONIT output record is requested for all tape volumes reserved “offline”.

(vsn,...)
The user transfers in a list (see below) the VSNs (volume serial numbers) of the tapes about which information is required.

addr
Symbolic address (name) of a field. The field contains a list of VSNs. It must be aligned on a word boundary.

(r)
Register containing the address value “addr”. A register may be specified only in conjunction with MF=M.
Format of the **list** for transferring the VSNs:

- The first word contains the number of entries in the list, right-justified in hexadecimal form.
- This is followed by the entries: each entry is 8 bytes long and contains (left-justified) the VSN. Bytes not required should be overwritten with X'00'.

Register contents

Register R15 is used internally as a general register when the **NKDINF** macro is called with MF=M and addresses are specified.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the NKDINF macro is transferred in the standard header: (cc=Subcode2, bb=Subcode1, aaaa=Maincode)

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function successfully executed. All requested information is provided in the output area
X'01' [1]	X'00'	X'0008'	Output area was created but not all requested output records are available: some or all of the specified codes (device type codes, VSNs, mnemonic names etc.) are unknown
X'02' [1]	X'00'	X'0008'	Output area was created but not all requested output records are available: user is not sufficiently privileged
X'04' [1]	X'00'	X'0008'	Output area was created but in some output records the information is not complete: disk monitor not available
X'08' [1]	X'00'	X'0008'	Output area was created but the information is incomplete in some output records: tape monitor not available
X'10' [1]	X'00'	X'0008'	Output area was created but the DRV operand could not be processed: DRV subsystem is not available
X'20' [1]	X'00'	X'0008'	Output area was created but the information is incomplete in some output records: four-digit mnemonic device names are available and EXTMN=NO was specified in the macro call. The 2-byte field in the affected output records was deleted.
X'xx' [2]	X'01'	X'0010'	Operand error, no output area was created: type of requested information is unknown or not permissible
X'xx' [2]	X'01'	X'0011'	Operand error, no output area was created: one of the outputs MN, VSN, DEV# etc. is unknown or not permissible
X'00'	X'01'	X'0013'	Operand error, no output area was created: EXTMN operand is unknown
X'00'	X'20'	X'0004'	System error, no output area was created
X'xx' [2]	X'40'	X'0012'	Type of a requested output record (record type) for the nonprivileged user is not allowed
X'xx' [2]	X'40'	X'0020'	List with the provided device type codes, VSNs, mnemonic names etc. has not been assigned
X'xx' [2]	X'40'	X'0021'	List with the provided device type codes, VSNs, mnemonic names etc. is not aligned on a word boundary

X'cc'	X'bb'	X'aaaa'	Meaning
X'xx' [2]	X'40'	X'0022'	More device type codes, VSNs, mnemonic names, etc. than allowed are specified in a list
X'xx' [2]	X'40'	X'0023'	Version number of a layout is unknown
X'00'	X'80'	X'0040'	No memory is currently available, no output area was created

[1] The sub return codes are added to the main code X'0008': when a macro is called several of them can occur simultaneously

[2] X'xx' indicates the type of output record (record type) which led to the return code. The possible values for xx are as follows:

01	TASK output record
02	GLOBAL output record
03	DEVICE output record
04	TAPE output record
05	DISC output record
06	CONFIG output record
07	DVQ output record
08	SUMMARY output record
09	TYPTASK output record
0A	UNMONIT output record
0B	DRV output record
0C	DEPOT output record
0D	LOC output record

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

The calling program is terminated when the following errors occur:
- The data area is not assigned to the caller.
- The data area is not aligned on a word boundary.
- The data area is protected against write access.

Layout of the DSECT for the output control record (RECORD=HEADER)

```
           NKDINF MF=D,RECORD=HEADER
1                *,NKDINF VERSION 500
1          #INTF INTNAME=NKDINF,REFTYPE=REQUEST,INTCOMP=1
1 *
1 *        GENERATION OF OUTPUT-LAYOUTS
1 *
1 *
1 *    C/DSECT FOR OUTPUT-CONTROL RECORD
1 *
1          MFCHK MF=D,PREFIX=N,SUPPORT=(C,D),MACID=KDI,                 C
1                DMACID=KDO,DNAME=KDOENT
2 NKDOENT  DSECT ,
2                *,##### PREFIX=N, MACID=KDO #####
1 NKDOENTR DS    OF               1 OUTPUT_CONTROL_RECORD
1 NKDOBLEN DS    F                  2 LENGTH_OF_BUFFER_OBTAINED
1 NKDOSBUF DS    OF                 2 SUBBUFFER_DESCRIPTION
1 NKDOTRBP DS    A                    3 PTR_TO_FIRST_TASK_RECORD
1 NKDOTRB# DS    H                    3 #_OF_TASK_RECORDS
1 NKDOTRBL DS    H                    3 LENGTH_OF_TASK_RECORD
1 NKDOTRBR DS    X                    3 RESULT_FOR_TASK_RECORDS SET
1 NKDOOKRO EQU   X'00'                  (OK_RECORDS_OUTPUTED
1 NKDOOKPT EQU   X'02'                   OK_RECORDS_PARTIAL_OUTPUT
1 NKDOOKRN EQU   X'04'                   OK_RECORDS_NOT_REQUESTED
1 NKDOOKRE EQU   X'08'                   OK_NO_RECORDS_EXISTS
1 NKDONOTA EQU   X'0C'                   SCOPE_HIDES_REQUESTED_RECORDS
1 NKDOOKMN EQU   X'0E'                   OK_NO_MN_DUE_TO_EXTENDED_MN
1 NKDODRBD DS    OF                 2 DEVICE_RECORD_BUFFER_DESCRIPTION
1 NKDODRBP DS    A                    3 PTR_TO_FIRST_DEVICE_RECORD
1 NKDODRB# DS    H                    3 #_OF_DEVICE_RECORDS
1 NKDODRBL DS    H                    3 LENGTH_OF_DEVICE_RECORD
1 NKDODRBR DS    X                    3 RESULT_FOR_DEVICE_RECORDS SET
1 * &P.OKRO  EQU   X'00'                  (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                   OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                   OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                   OK_NO_RECORDS_EXISTS
1 * &P.NOTA  EQU   X'0C'                   SCOPE_HIDES_REQUESTED_RECORDS
1 * &P.OKMN  EQU   X'0E'                   OK_NO_MN_DUE_TO_EXTENDED_MN
1 NKDOGRBD DS    OF                 2 GLOBAL_RECORD_BUFFER_DESCRIPTION
1 NKDOGRBP DS    A                    3 PTR_TO_GLOBAL_RECORD
1 NKDOGRB# DS    H                    3 #_OF_GLOBAL_RECORDS "ALWAYS 1"
1 NKDOGRBL DS    H                    3 LENGTH_OF_GLOBAL_RECORD
1 NKDOGRBR DS    X                    3 RESULT_FOR_GLOBAL_RECORD SET
1 * &P.OKRO  EQU   X'00'                  (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                   OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                   OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                   OK_NO_RECORDS_EXISTS
```

```
1 * &P.NOTA  EQU   X'0C'                  SCOPE_HIDES_REQUESTED_RECORDS
1 NKDOCRBD DS    0F                2 CONFIG_RECORD_BUFFER_DESCRIPTION
1 NKDOCRBP DS    A                   3 PTR_TO_FIRST_CONFIG_RECORD
1 NKDOCRB# DS    H                   3 #_OF_CONFIG_RECORDS
1 NKDOCRBL DS    H                   3 LENGTH_OF_CONFIG_RECORD
1 NKDOCRBR DS    X                   3 RESULT_FOR_CONFIG_RECORDS SET
1 * &P.OKRO  EQU   X'00'                 (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                  OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                  OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                  OK_NO_RECORDS_EXISTS
1 * &P.NOTA  EQU   X'0C'                  SCOPE_HIDES_REQUESTED_RECORDS
1 * &P.OKMN  EQU   X'0E'                  OK_NO_MN_DUE_TO_EXTENDED_MN
1 NKDOBRBD DS    0F               2 TAPE_RECORD_BUFFER_DESCRIPTION
1 NKDOBRBP DS    A                  3 PTR_TO_FIRST_TAPE_RECORD
1 NKDOBRB# DS    H                  3 #_OF_TAPE_RECORDS
1 NKDOBRBL DS    H                  3 LENGTH_OF_TAPE_RECORD
1 NKDOBRBR DS    X                  3 RESULT_FOR_TAPE_RECORDS SET
1 * &P.OKRO  EQU   X'00'                 (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                  OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                  OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                  OK_NO_RECORDS_EXISTS
1 * &P.NOTA  EQU   X'0C'                  SCOPE_HIDES_REQUESTED_RECORDS
1 * &P.OKMN  EQU   X'0E'                  OK_NO_MN_DUE_TO_EXTENDED_MN
1 NKDOPRBD DS    0F               2 DISC_RECORD_BUFFER_DESCRIPTION
1 NKDOPRBP DS    A                  3 PTR_TO_FIRST_DISC_RECORD
1 NKDOPRB# DS    H                  3 #_OF_DISC_RECORDS
1 NKDOPRBL DS    H                  3 LENGTH_OF_DISC_RECORD
1 NKDOPRBR DS    X                  3 RESULT_FOR_DISC_RECORDS SET
1 * &P.OKRO  EQU   X'00'                 (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                  OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                  OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                  OK_NO_RECORDS_EXISTS
1 * &P.NOTA  EQU   X'0C'                  SCOPE_HIDES_REQUESTED_RECORDS
1 * &P.OKMN  EQU   X'0E'                  OK_NO_MN_DUE_TO_EXTENDED_MN
1 NKDOQRBD DS    0F                2 DVQ_RECORD_BUFFER_DESCRIPTION
1 NKDOQRBP DS    A                   3 PTR_TO_DVQ_RECORD
1 NKDOQRB# DS    H                   3 #_OF_DVQ_RECORDS "ALWAYS 1"
1 NKDOQRBL DS    H                   3 LENGTH_OF_DVQ_RECORD
1 NKDOQRBR DS    X                   3 RESULT_FOR_DVQ_RECORD SET
1 * &P.OKRO  EQU   X'00'                 (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                  OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                  OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                  OK_NO_RECORDS_EXISTS
1 * &P.NOTA  EQU   X'0C'                  SCOPE_HIDES_REQUESTED_RECORDS
1 * &P.OKMN  EQU   X'0E'                  OK_NO_MN_DUE_TO_EXTENDED_MN
1 NKDOSRBD DS    0F                2 SUMMARY_RECORD_BUFFER_DESCRIPTION
1 NKDOSRBP DS    A                   3 PTR_TO_SUMMARY_RECORD
1 NKDOSRB# DS    H                   3 #_OF_SUMMARY_RECORDS
```

```
1 NKDOSRBL DS    H                   3 LENGTH_OF_SUMMARY_RECORD
1 NKDOSRBR DS    X                   3 RESULT_FOR_SUMMARY_RECORD SET
1 * &P.OKRO  EQU   X'00'                 (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                  OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                  OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                  OK_NO_RECORDS_EXISTS
1 * &P.NOTA  EQU   X'0C'                  SCOPE_HIDES_REQUESTED_RECORDS
1 NKDOYRBD DS    OF                2 TYPTASK_RECORD_BUFFER_DESCRIPTION
1 NKDOYRBP DS    A                   3 PTR_TO_TYPTASK_RECORD
1 NKDOYRB# DS    H                   3 #_OF_TYPTASK_RECORDS
1 NKDOYRBL DS    H                   3 LENGTH_OF_TYPTASK_RECORD
1 NKDOYRBR DS    X                   3 RESULT_FOR_TYPTASK_RECORD SET
1 * &P.OKRO  EQU   X'00'                 (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                  OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                  OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                  OK_NO_RECORDS_EXISTS
1 * &P.NOTA  EQU   X'0C'                  SCOPE_HIDES_REQUESTED_RECORDS

1 NKDOURBD DS    OF                2 UNMONIT_RECORD_BUFFER_DESCRIPTION

1 NKDOURBP DS    A                   3 PTR_TO_UNMONIT_RECORD
1 NKDOURB# DS    H                   3 #_OF_UNMONIT_RECORDS
1 NKDOURBL DS    H                   3 LENGTH_OF_UNMONIT_RECORD
1 NKDOURBR DS    X                   3 RESULT_FOR_UNMONIT_RECORD SET
1 * &P.OKRO  EQU   X'00'                 (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                  OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                  OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                  OK_NO_RECORDS_EXISTS
1 * &P.NOTA  EQU   X'0C'                  SCOPE_HIDES_REQUESTED_RECORDS
1 NKDOVRBD DS    OF                2 DRV_RECORD_BUFFER_DESCRIPTION
1 NKDOVRBP DS    A                   3 PTR_TO_DRV_RECORD
1 NKDOVRB# DS    H                   3 #_OF_DRV_RECORDS
1 NKDOVRBL DS    H                   3 LENGTH_OF_DRV_RECORD
1 NKDOVRBR DS    X                   3 RESULT_FOR_DRV_RECORD SET
1 * &P.OKRO  EQU   X'00'                 (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                  OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                  OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                  OK_NO_RECORDS_EXISTS
1 * &P.NOTA  EQU   X'0C'                  SCOPE_HIDES_REQUESTED_RECORDS
1 NKDOLRBD DS    OF                2 DEPOT_RECORD_BUFFER_DESCRIPTION
1 NKDOLRBP DS    A                   3 PTR_TO_DEPOT_RECORD
1 NKDOLRB# DS    H                   3 #_OF_DEPOT_RECORDS
1 NKDOLRBL DS    H                   3 LENGTH_OF_DEPOT_RECORD
1 NKDOLRBR DS    X                   3 RESULT_FOR_DEPOT_RECORD SET
1 * &P.OKRO  EQU   X'00'                 (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                  OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                  OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                  OK_NO_RECORDS_EXISTS
```

```
1 * &P.NOTA  EQU   X'0C'                    SCOPE_HIDES_REQUESTED_RECORDS
1 NKDOARBD DS    0F                  2 LOCATION_REC_BUFFER_DESCRIPTION
1 NKDOARBP DS    A                     3 PTR_TO_LOCATION_RECORD
1 NKDOARB# DS    H                     3 #_OF_LOCATION_RECORDS
1 NKDOARBL DS    H                     3 LENGTH_OF_LOCATION_RECORD
1 NKDOARBR DS    X                     3 RESULT_FOR_LOCATION_RECORD SET
1 * &P.OKRO  EQU   X'00'                   (OK_RECORDS_OUTPUTED
1 * &P.OKPT  EQU   X'02'                    OK_RECORDS_PARTIAL_OUTPUT
1 * &P.OKRN  EQU   X'04'                    OK_RECORDS_NOT_REQUESTED
1 * &P.OKRE  EQU   X'08'                    OK_NO_RECORDS_EXISTS
1 * &P.NOTA  EQU   X'0C'                    SCOPE_HIDES_REQUESTED_RECORDS
1          DS    XL3               RESERVED
1 NKDO#    EQU   *-NKDOENTR        LENGTH_OF_CONTROL_RECORD
1 NKDOLGTH EQU   NKDO#             OLD NAME OF LENGTH
```

NKGTYPE – Output device information

General

Application area: Requesting and accessing lists and tables; see page 155
Macro type: Type S, MF format **1**:
31-bit interface: standard/E/L/D form;
see page 29

Device type: Each device is assigned to a device type (DEVICE type). The devices of a particular device type have the same device characteristics. The device type can be addressed either by its (printable) name or by a device type code. Examples of device type names are HNC2, D3435; the corresponding device type codes are X'6D', X'A5'.

Volume type: The volume type defines a combination of (device) characteristics which are important for the use of a volume, e.g. the recording density of tapes. Those device types which possess the characteristics required of a particular type of volume are assigned to a volume type; e.g. the device type LTO-U4 is assigned to volume type TAPE-U4 (MTC with 896 tracks). For disks, the assignment of device types to volume types is unique (identical names). The volume type can be addressed either by its (printable) name or by the volume type code. An examples of name is TAPE-U4; volume type code X'CE'.

Device family: A set of device types having specific device characteristics are assigned to a device family (FAMILY type). A given device type will belong to one device family only. A device family can be addressed by its (printable) name or by the family code. Examples of device family names are DISK, MTC; the corresponding family codes are X'A0', X'C0'.

Device class: A set of device types is assigned to a device class (CLASS type). Three device classes exist: UR, TAPE and DISK (where UR stands for UNIT-RECORD devices). Device classes can be addressed by their (printable) names only.

The device type table (“System Installation” [10] manual) contains the names of device types and device families and the associated device type codes and family codes.

Macro description

The **NKGTYPE** macro provides information about the names, device type codes, device characteristics, path addresses and path attributes for a particular device type or volume type, or about names and device type codes of the device types of a particular device family or device class.
The scope of the information to be output can be determined by means of the INF=... operand. The operand list starts with the standard header. The user can request a DSECT of the input/output area to be generated. The size of the DSECT is also determined by means of the INF operand.

Macro format and description of operands

NKGTYPE
[p] [,MF=(E,..) / L / D] ,FORMAT=INTERNAL / PRINTABLE ,TYPE=UNKNOWN / DEVICE / VOLUME / FAMILY / CLASS [,INTYP=addr] ,INF=STD / LIST / LOCLIST / VLIST / GEN / STD-LIST

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

If MF=D is specified, a prefix (p = 1 letter) can be specified, as shown in the macro format.

If the MF operand is omitted, parameter list and calling sequence are generated. After execution R1 then points to the parameter list.

FORMAT=
Indicates whether the name or the code is specified.

INTERNAL
The code (device type code/volume type code/family code) is specified.

PRINTABLE
The (printable) name is specified.

TYPE=
Indicates the device type, volume type, device family or device class to which the device/device type belongs.

UNKNOWN
The assignment is unknown. The type lists are searched in the following order: device class, device family, volume type and device type (including the lists of the corresponding “alias”). The list of device classes is searched only if the (printable) name is specified.

DEVICE
Only the list of device types and the list of the corresponding “alias” is searched.

VOLUME
Only the list of volume types and the list of the corresponding “alias” is searched.

FAMILY
Only the list of device families is searched.

CLASS
Only the list of device classes is searched.

INTYP=
Specifies the field containing the name or code. If the INTYP operand is omitted, the input field NKGDIPTI/NKGDIPPR in the operand list must be supplied with the name/code by the user. Field length:

- 8 bytes if the (printable) name is specified. The name is entered left-justified with trailing blanks.
- 2 bytes if the device type code or family code is specified. The code is entered left-justified with trailing zeros.

addr
Symbolic address of the field.

INF=
Defines the scope of the information output.

STD
Standard output is generated:

- type list which contains the name or code
- device class (CLASS type)
- name and code
- If a device type is specified, a list of the device type characteristics (family code, recording density of a magnetic tape device, ...) is additionally output.

LIST
In addition to the standard output, a list of the corresponding device types is output:

– If a device type is specified, a list of the corresponding volume types is output and vice versa.
– If a device family or device class is specified, a list of the names and device type codes of the device types belonging to that device family or device class is output.

LOCLIST
In addition to the standard output, a list of the corresponding types in the specified storage location is also output. The name of the storage location must be supplied explicitly in the parameter list. If no value is entered in the storage location field, the corresponding types are determined from the set of devices which are not assigned to any storage location (residual pool).
You can only specify a device type or a volume type.
If you specify a device type (TYPE=DEVICE), you will obtain a list of the corresponding volume types; if you specify a volume type (TYPE=VOLUME), you will see a list of the corresponding device types.

VLIST
Specification of this operand is meaningful for device classes only.
In addition to the standard output, a list of the volume types belonging to the device class specified is output.

GEN
Specification of this operand is meaningful for device types only.
In addition to the standard output, a list of the generation options (path addresses and path attributes, e.g. channel type, controller type, ...) is output.

STD-LIST
Specification of this operand is meaningful for TYPE=CLASS only.
In contrast to INF=LIST, only the list of disk devices and STDDISK supported to BS2000/OSD-BC V2.0 is output.
This value was introduced to support the SDF data type <device>. New disks should only be requested in user commands with the standard disk type `STDDISK`.

Return information and error flags

Register R1 contains the operand list address.

R15:	c	c	b	b	a	a	a	a

A structured return code relating to the execution of the NKGTYPE macro is transferred in register R15:
(cc=Subcode2, bb=Subcode1, aaaa=Maincode)

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Normal execution, entry found
X'00'	X'04'	X'0004'	Entry not found
X'00'	X'08'	X'0004'	Specification for INF=... is inconsistent with specification for TYPE=...
X'00'	X'10'	X'0004'	The (printable) name specified is incomplete or ambiguous
X'00'	X'20'	X'0004'	System error
X'00'	X'0C'	X'0004'	Invalid operand list address
X'00'	X'FF'	X'0004'	Operand error

Layout of the DSECT for the input/output area if INF=STD

```
           NKGTYPE MF=D
1                *,VERSION 701
1          #INTF MACNAME=NKGTYPE,REFTYPE=REQUEST,MACVERS=701,            C
1                INTNAME=011.001,INTVERS=1
1          MFPRE DNAME=KGDTYP,MF=D,MACID=KGD,ALIGN=F,PREFIX=N,           C
1                DMACID=KGD
2 NKGDTYP  DSECT ,
2                *,##### PREFIX=N, MACID=KGD #####
1 NKGDPLS  DS    0F       DCL 1 NKGTYPE_PARAMETERLIST_START
1 NKGDFHDR FHDR  MF=(C,NKGD),EQUATES=NO
2 NKGDFHDR DS    0A
2 NKGDFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 NKGDIFID DS    0A              0   INTERFACE IDENTIFIER
2 NKGDFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 NKGDFCT  DS    AL1             2   FUNCTION NUMBER
2 NKGDFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
```

```
2 NKGDRET  DS    OA              4   GENERAL RETURN CODE
2 NKGDSRET DS    OAL2            4   SUB RETURN CODE
2 NKGDSR2  DS    AL1             4   SUB RETURN CODE 2
2 NKGDSR1  DS    AL1             5   SUB RETURN CODE 1
2 NKGDMRET DS    OAL2            6   MAIN RETURN CODE
2 NKGDMR2  DS    AL1             6   MAIN RETURN CODE 2
2 NKGDMR1  DS    AL1             7   MAIN RETURN CODE 1
2 NKGDFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1          SPACE 1
1 *        EQUATES FOR THE MAIN RETURN CODE
1          SPACE 1
1 NKGDSUCC EQU   X'0000'          SUCCESS : ENTRY FOUND
1 *                               OUTPUT AREA CONTAINS VALID DATA
1 NKGDPERR EQU   X'0004'          UNSUCCESS
1 *                               OUTPUTAREA IS SET TO ZERO (X'00')
1 *                               SEE SUB RETURN CODE 1 FOR DETAIL
1          SPACE 1
1 *        EQUATES FOR THE SUB RETURN CODE 1 (RR IN X'00RR0004)
1          SPACE 1
1 NKGDSENF EQU   X'04'            ENTRY NOT FOUND
1 NKGDICIT EQU   X'08'            UNCONSISTENCY BETWEEN INF AND TYPE P.
1 NKGDS1SY EQU   X'20'            SYSTEM ERROR   (RC-CLASS C)         215
1 NKGDSPIA EQU   X'10'            PRINTABLE INPUT IS AMBIGUOUS
1 NKGDSPER EQU   X'FF'            PARAMETER ERROR
1 NKGDNAP1 EQU   X'0C'            PARAMETER LIST OR INPUT-TYPE NOT
1 *                               ACCESSIBLE (THIS RC IS ONLY
1 *                               POSSIBLE FOR P1-USERS, FOR WHICH A
1 *                               ADDRESS VALIDATION IS DONE, THIS RC
1 *                               IS ONLY SET TO CALLERS R15)
1          SPACE 1
1 *        EQUATES FOR THE SUB RETURN CODE 2  (SS IN X'SS000000)      101
1          SPACE 1
1 *********************************
1 *        INPUT PARAMETER        *
1 *********************************
1 NKGDINP  DS    OF        2 INPUT_PARAMETER
1 NKGDTPES DS    X           3 TYPE_SET
1 NKGDDEVC EQU   C'D'          (DEVICE  = D,
1 NKGDVOLM EQU   C'V'           VOLUME  = V,
1 NKGDFAML EQU   C'F'           FAMILY  = F,
1 NKGDCLSS EQU   C'C'           CLASS   = C,
1 NKGDUNKN EQU   C'U'           UNKNOWN = U)
1 NKGDFRMT DS    X           3 FORMAT_SET
1 NKGDINTR EQU   X'01'         (INTERNAL  = 1,
1 NKGDPRNT EQU   X'02'          PRINTABLE = 2)
```

```
1 NKGDTYPE DS    X           3 INPUT_TYPE_SET
1 NKGDTSTD EQU   X'00'         (TYPE_EXPLIZIT_IN_PARAMLIST   = 0,
1 NKGDADDR EQU   X'01'          TYPE_ADDR_SPECIFIED          = 1,
1 NKGDREG  EQU   X'02'          NO MORE USED SINCE V11.2A )
1 NKGDINFS DS    X           3 INFORMATION_SET
1 NKGDISTD EQU   X'01'         (STANDARD                     = 1,
1 NKGDIGEN EQU   X'02'          GENERATION                   = 2,
1 NKGDILST EQU   X'04'          LIST_OF_CORRESPONDING_TYPES  = 4,
1 *I.ICHR  EQU   X'08'          DISK_CHARACTERISTICS NO LONGER SUPP.
1 NKGDVLST EQU   X'10'          LIST_OF_CLASS_SPEC_VOL_TYPES = 16,
1 NKGDSLST EQU   X'20'          Standard_List_of_CLASS       = 32,
1 NKGDLLST EQU   X'40'          LIST-OF-CORR-TYPES-IN-LOCATION=64 )
1          SPACE 1
1 NKGDIADR DS    A           3 ADDR(INPUT_TYPE)
1          ORG   NKGDIADR
1 NKGDIREG DS    A           3 NO MORE USED SINCE V11.2A
1 NKGDIPTI DS    XL2         3 INPUT_TYPE_INTERNAL_FORMAT
1          DS    0A
1 NKGDIPPR DS    CL8         3 INPUT_TYPE_PRINTABLE_FORMAT
1 NKGDILOC DS    CL8         3 INPUT_LOCATION
1          SPACE 2
1 *********************************
1 *        OUTPUT PARAMETER       *
1 *********************************
1 NKGDOUTP DS    0F        2 OUTPUT_PARAMETER
1 NKGDOPLI DS    CL1         3 OUTPUT_LIST_INDICATOR SET
1 NKGDOTTL EQU   C'D'          (ENTRY_FOUND_IN_DEVICE_TYPE_LIST=D,
1 NKGDOVTL EQU   C'V'           ENTRY_FOUND_IN_VOLUME_TYPE_LIST=V,
1 NKGDOFTL EQU   C'F'           ENTRY_FOUND_IN_FAMILY_TYPE_LIST=F,
1 NKGDOCTL EQU   C'C'           ENTRY_FOUND_IN_CLASS_TYPE_LIST =C)
1 NKGDSECL DS    XL1         3 SECOND_LIST_INDICATOR
1 NKGDSVTL EQU   X'01'         4 E._FOUND_ALSO_IN_VOL_TYPE_L = 1
1 NKGDSDTL EQU   X'02'         4 E._FOUND_ALSO_IN_DEV_TYPE_L = 2
1 NKGDSFML EQU   X'04'         4 E._FOUND_ALSO_IN_FAM_TYPE_L = 4
1 NKGDOPNI DS    XL1         3 NAME_INDICATOR
1 *                             /* TO TEST ON BIT LEVEL */
1 NKGDOINS EQU   X'01'         4 INPUT_NAME_IN_SHORT_FORMAT = 1
1 NKGDOANT EQU   X'02'         4 ALIAS_NAME_TRANSLATED      = 2
1 NKGDDCLS DS    X           3 DEVICE_CLASS SET
1 NKGDURD  EQU   C'U'            (UR_DEVICE  =U,
1 NKGDDISC EQU   C'D'             DISC_DEVICE=D,
1 NKGDTAPE EQU   C'T'             TAPE_DEVICE=T)
1          DS    X           3 RESERVED
1 NKGDDVTI DS    XL2         3 TYPE_INTERNAL_FORMAT
1 NKGDDVTP DS    CL8         3 TYPE_PRINTABLE_FORMAT
```

```
1 NKGDRESP DS    X           3 RESERVATION_POSSIBILITY_INDICATOR
1 NKGDRESN EQU   X'01'         4 RESERVATION_BY_TYPE_NOT_POSSIBLE = 1
1 NKGDALNP EQU   X'02'         4 ALLOCATION_BY_TYPE_NOT_POSSIBLE  = 2
1 NKGDRMNN EQU   X'04'         4 RESERVATION_BY_MNEM_NOT_POSSIBLE = 4
1 NKGDAMNN EQU   X'08'         4 ALLOCATION_BY_MNEM_NOT_POSSIBLE  = 8
1 NKGDATNA EQU   X'10'         4 ATTACH_NOT_ALLOWED               =16
1          DS    F           3 RESERVED
1          SPACE 2
1 ***********************************************************
1 *
1 *   VOLUME-TYPE-ATTRIBUTES
1 *   ONLY VALID IF NKGDOPLI=NKGDOVTL
1 *   (ENTRY_FOUND_IN_VOLUME_TYPE_LIST_
1 *
1 ***********************************************************
1 NKGDDNSF DS    X           3 DENSITY_FLAG
1 NKGDCORD EQU   X'01'         4 VOL_TYPE_CORRESPONDS_TO_ONE_DENSITY
1 NKGDVLEV DS    X           3 TAPE_LEVEL_INDICATOR
1 NKGDLV9  EQU   X'01'         4 9_LEVEL_TAPE
1 NKGDLV18 EQU   X'02'         4 MBK_Mode
1 NKGDLVV  EQU   X'04'         4 VIDEO_TAPE
1 NKGDLVSC EQU   X'08'         4 STREAMING_CARTRIGE_TAPE
1 NKGDITYP DS    XL2         3 VOLUME_TYPE_FOR_LABEL_INITIALISATION
1 NKGDVL#2 EQU   *             VOLUME_TYPE_DESCRIPTION_EXTENSION
1 NKGDVRCM EQU   *           3 VOLUME_TYPE_RECORDING_MODE
1 NKGDVTPM DS    X             4  VOLUME_TAPE_MODE ( /* 9_LEVEL_TAPE*/
1 *I.800   EQU   X'01'              DENSITY_800_BPI  = 1,
1 *I1600   EQU   X'02'              DENSITY_1600_BPI = 2,
1 *I6250   EQU   X'04'              DENSITY_6250_BPI = 4 )
1          ORG   NKGDVTPM
1 NKGDV18M DS    X            4   VOLUME_MBK_MODE  (/* 18_LEVEL_TAPE*/
1 *I.COMP  EQU   X'01'              DATA_COMPACTION        = 1
1 *I.256P  EQU   X'02'              256kB_blocks_possible  = 2
1 *I.MLTE  EQU   X'20'              LTO_encrypted          = 32
1 *I.MLTO  EQU   X'40'              MODE LTO               = 64 )
1 NKGDWORN EQU   X'80'              WORM-Medium            = 128 )
1 NKGDTRCT DS    CL2        3 DMS_TAPE_RECORDING_TECHNIQUE(
1 NKGDDLNR EQU   C'  '              NORMAL_DENSITY       ,
1 NKGDDLCM EQU   C'P '              COMPACTION           )
1 NKGDDHDI DS    C          3 DMS_HDR2_DENSITY_INDICATOR(
1 NKGDDH8  EQU   C'2'               HDR2_800_BPI         ,
1 NKGDDH16 EQU   C'3'               HDR2_1600_BPI        ,
1 NKGDDH62 EQU   C'4'               HDR2_6250_BPI        ,
1 NKGDDHDM EQU   C' '               HDR2_MBK             )
1 NKGDTCRS DS    X          3 TYPE_CONCERNED_BY_RESTRICTION
1 NKGDRCNW EQU   X'01'         CONCERNED_BY_RESTRICTION_NO_WRITE
1 NKGDLTOT DS    X          3 LTO_MEDIUM_TYPE
1          ORG   NKGDVL#2
```

```
1          SPACE 2
1 ***********************************************************
1 *
1 *   GENERATION-UNDEPENDENT DEVICE-TYPE-ATTRIBUTES
1 *       ONLY VALID IF NKGDOPLI=NKGDOTTL
1 *       (ENTRY_FOUND_IN_DEVICE_TYPE_LIST)
1 *
1 ***********************************************************
1 NKGDFAMD DS    0XL10       3 FAMILY_DESCRIPTION
1 NKGDFAMI DS    XL2           4 FAMILY_INTERNAL_FORMAT
1 NKGDFAMP DS    CL8           4 FAMILY_PRINTABLE_FORMAT
1 NKGDDEFI DS    X           3 DEFAULT_INTERRUPTION_SUBCLASS_CODE_SET
1 NKGDDEF0 EQU   X'00'
1 NKGDDEF1 EQU   X'01'
1 NKGDDEF2 EQU   X'02'
1 NKGDDEF3 EQU   X'03'
1 NKGDDEF4 EQU   X'04'
1 NKGDDEF5 EQU   X'05'
1 NKGDDEF6 EQU   X'06'
1 NKGDDEF7 EQU   X'07'
1 NKGDHWA2 DS    X           3 HARDWARE_ATTRIBUTES BYTE 2
1 *        EQUATES  CF BELOW
1 NKGDDSSP DS    0XL2        3 DISC_SUB_ATTRIBUTES
1 NKGDDMDE DS    X             4 DISC_MODE
1 NKGDMOVJ EQU   X'01'           5 MOVABLE  =1
1 NKGDFIXH EQU   X'02'           5 FIX_HEAD =2
1 NKGDSTRU DS    X             4 DISK_STRUCTUR_SET
1 NKGDSFBA EQU   X'01'           (FBA-DISK          =1,
1 NKGDSCKD EQU   X'02'            CKD-DISK          =2,
1 NKGDSPAD EQU   X'04'            PAD_DISK          =4,
1 NKGDSCHD EQU   X'08'            CHANNEL_DEPENDENT =8, &CTL-DEPENDENT
1 NKGDSECK EQU   X'10'            ECKD_DISK         =16,
1 NKGDSEMU EQU   X'20'            EMULATED_DISK     =32)
1          ORG   NKGDDSSP
1 NKGDTSSP EQU   *           3 TAPE_SUB_ATTRIBUTES
1 NKGDTMDE DS    X             4 TAPE_MODE  /* 9_LEVEL_TAPE */
1 NKGDL9M  EQU   NKGDTMDE
1 NKGD800  EQU   X'01'           5 No longer used
1 NKGD1600 EQU   X'02'           5 1600BPI_SUPPORTED        =2
1 NKGD6250 EQU   X'04'           5 6250BPI_SUPPORTED        =4
1          ORG   NKGDTMDE
1 NKGDL18M DS    X             4 MBK_MODE
1 NKGDCOMP EQU   X'01'           5 DATA_COMPATCION_SUPPORTED = 1
1 NKGDTPLV DS    X           3 TAPE_LEVEL_INDICATOR
1 *&I.LV9  EQU   X'01'         4 9_LEVEL_TAPE              = 1
1 *&I.LV18 EQU   X'02'         4 MBK_Mode                  = 2
1 *&I.LVV  EQU   X'04'         4 VIDEO_TAPE
1 *&I.LVSC EQU   X'08'         4 STREAMING_CARTRIGE_TAPE
```

```
1          ORG   NKGDDSSP
1 NKGDCNSP EQU   *           3 CONSOLE_SUB_ATTRIBUTES
1 NKGDCMDE DS    X             4 CONSOLE_MODE
1 NKGDCDIS EQU   X'01'           5 DISPLAY =1
1 NKGDCHDC EQU   X'02'           5 HARDCOPY=2
1          ORG   NKGDDSSP+L'NKGDDSSP
1 NKGDHWAT DS    X          3 HARDWARE_ATTRIBUTES
1 NKGDRERE EQU   X'01'        4 RESERVE_RELEASE_SUPPORTED = 1  (D)
1 NKGDSDNC EQU   X'02'        4 SET_DENSITY_NECCESSARY    = 2  (T)
1 NKGDSTRM EQU   X'04'        4 STREAMING_MODUS_POSSIBLE  = 4  (T)
1 NKGDDISP EQU   X'08'        4 DISPLAY                   = 8  (T,D)
1 NKGDBUFF EQU   X'10'        4 CONTROLLER_WITH_BUFFER    =16  (T,D)
1 NKGDPOSI EQU   X'20'        4 CTL_SUPPORTS_POSITIONING  =32  (T)
1 NKGDNRRV EQU   X'40'        4 NO_READ_REVERSE_SUPPORTED =64  (T)
1 NKGDSTKP EQU   X'80'        4 STACKER_POSSIBLE          =128 (T)
1 *  HWA2                   3 HARDWARE_ATTRIBUTES BYTE 2
1 NKGDSSDD EQU   X'01'        4 SOLID_STATE_DISC_DEVICE   = 1  (D)
1 NKGDRSTP EQU   X'02'        4 RANDOM_STACKER_POSSIBLE   = 2  (T)
1 NKGDFORI EQU   X'04'        4 FORMATTING_DURING_INIT_NECESSARY= 4(T)
1 NKGDGENF DS    X          3 GENERATION_FACILITIES
1 NKGDSICH EQU   X'01'        4 CFCS12_CHANNEL_TYPE =1
1 NKGDFJCH EQU   X'02'        4 CFCS3_CHANNEL_TYPE  =2
1 NKGDPADT DS    A          3 ADRESS_OF_ADAM_TRANSLATION
1 NKGDCBUF DS    F          3 CONTROLLER_BUFFER_SIZE (BYTES)
1 NKGDDCAP DS    F          3 MAXIMAL DISC CAPACITY (#HP'S PER VOL)
1 NKGD#CYL DS    H          3 MAXILAL NUMBER OF CYCLINDERS PER VOL
1 NKGDDFVT DS    XL2        3 DEFAULT_VOLUME_TYPE (DEFAULT_DENSITY)
1 NKGDURST DS    X          3 USE_RESTRICTION
1 NKGDRNWR EQU   X'01'        RESTRICTION_NO_WRITE_POSSIBLE
1 NKGDLTOL DS    X          3 LTO_LEVEL
1          DS    XL2        3 RESERVED
1 NKGDOPL1 EQU   *-NKGDOUTP
1 NKGDOPLN EQU   *-NKGDOUTP  LENGTH_OF_THE_OUTPUT_PARAMETERLIST
1          DS    0XL(1-(NKGDOPLN-60)*(NKGDOPLN-60))
1 NKGDPLLN EQU   *-NKGDPLS   TOTAL_LENGTH_OF_THE_PARAMETERLIST
```

Example

The device types belonging to device family X'A0' (disk storage units) are to be output. The macro is called in the E/L form.

```
NKGTYPE  START
         PRINT NOGEN
         BALR  3,0
         USING *,3
         NKGTYPE MF=(E,NKGL)
END      TERM
NKGL     NKGTYPE MF=L,TYPE=FAMILY,INTYP=ADDR,INF=LIST
ADDR      DC    X'A000' ----------------------------------------------- (1)
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,nkgtype), -
//       compiler-action=module-generation(module-format=llm), -
//       module-library=macexmp.lib, -
//       listing=parameters(output=*library-element(macexmp.lib,nkgtype)), -
//       test-support=*aid
%  ASS6011 ASSEMBLY TIME: 306 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 94 MSEC
//end
%  ASS6012 END OF ASSEMBH
/load-executable-program library=macexmp.lib,element-or-symbol=nkgtype, -
/     test-options=*aid
%  BLS0523 ELEMENT 'NKGTYPE', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'NKGTYPE', VERSION ' ' OF '<date> <time>' LOADED
/%on %any <%d %§(nkgl)-> %xl568>;%r ------------------------------------ (2)
*** TID: 00710070 *** TSN: 6LHZ *****************************************
CURRENT PC: 0000002C    CSECT: NKGTYPE  *************************************
V'00000030' = NKGTYPE  + #'00000030'
00000030 (00000030) 000B0101 00000000 C6010104 000002BA    ........F.......
00000040 (00000040) 00000000 00000000 00000000 00000000    ................
00000050 (00000050) 00000000 C60000C4 00A000C4 C9E2D240    ....F..D...DISK
00000060 (00000060) 40404003 00000000 00000000 00000000       .............
00000070 (00000070) 00000000 00000000 00000000 00000000    ................
00000080 (00000080) 00000000 00000000 00000000 00000000    ................
00000090 (00000090) 001A0000 8100C4F3 F4F3F460 F1F08200    ....a.D3434-10b.
000000A0 (000000A0) C4F3F4F3 F460F2F0 8300C4F3 F4F3F860    D3434-20c.D3438-
```

```
000000B0 (000000B0) F3F08400 C4F3F4F2 F1F160F4 8500C4F3    30d.D34211-4e.D3
000000C0 (000000C0) F4F0F940 40408600 C4F3F4F2 F1F160F2    409   f.D34211-2
000000D0 (000000D0) 8700C4F3 F4F3F460 F3F08800 C4F3F4F2    g.D3434-30h.D342
000000E0 (000000E0) F1F160F3 8900C4F3 F4F9F060 F3F08A00    11-3i.D3490-30..
000000F0 (000000F0) C4F3F4F9 F060F4F0 8B00C4F3 F4F2F1F1    D3490-40..D34211
00000100 (00000100) 60F58E00 C4F3F4F9 F2404040 8F00C4F3    -5..D3492   ..D3
00000110 (00000110) F4F7F560 F8C6A100 C4F3F4F3 F960F1F0    475-8F..D3439-10
00000120 (00000120) A200C4F3 F4F3F640 4040A300 C4F3F4F3    s.D3436   t.D343
00000130 (00000130) F7404040 A400C4F3 F4F3F860 F2F0A500    7   u.D3438-20v.
00000140 (00000140) C4F3F4F3 F5404040 A700C4F3 F4F9F060    D3435   x.D3490-
00000150 (00000150) F1F0AAF0 C4F3F4F0 F960C7E2 AB00C4F3    10.0D3409-GS..D3
00000160 (00000160) F4F7F540 4040AC00 C4F3F4F8 F0404040    475   ..D3480
00000170 (00000170) AD00C4F3 F4F8C540 4040AE00 C4F3F4F8    ..D348E   ..D348
00000180 (00000180) C6404040 AF00C4F3 F4F9F060 F2F0AA00    F   ..D3490-20..
00000190 (00000190) E2E3C4C4 C9E2D240 00000000 00000000    STDDISK ........
000001A0 (000001A0) 00000000 00000000 00000000 00000000    ................
          REPEATED LINES:    10
00000250 (00000250) 00000000 00000000 00000000 00000000    ................
00000260 (00000260) 00000000 00000000                      ........
```

(1) The family code for disk storage units is entered.

(2) The input output area is displayed using the Advanced Interactive Debugger (AID).

The I/O area starts at address X'000034'.
Bytes 3-7 contain the return code X'00000000'.
The input data (DSECT area “INPUT PARAMETER”) follows:
X'C6': TYPE=FAMILY,
X'01': FORMAT=INTERNAL,
X'01': INTYP=ADR,
X'04': INF=LIST,
X'000002BA' is the virtual address of field ADR.

The output area (DSECT area “OUTPUT PARAMETER”) starts at address X'000054', where:
X'C6': ,Entry was found in the list of device families.
X'C4': ,Disk storage unit.
X'A0': ,Family code.
X'C4C9E2D240404040': DISK = name of the device family.

The additional output for INF=LIST is found starting at address X'000090'.
X'001A': The device type list contains 26 entries.
2 reserved bytes are followed by the device type code and then the (printable) device type name.

NSIINF – Output system information

General

Application area: Requesting and accessing lists and tables; see page 155
Macro type: Type S, MF format **3**: D/C/E/L form; see page 29

Macro description

The **NSIINF** macro provides information about the following:

CPU: serial numbers and identifications of available CPUs
BS2000: label and version of the operating system, as well as addressing mode of the operating system and the operating system options selected at generation time
Memory: size of the (physical) main memory
HSI: HSI base type
Server: type of server (model series)
Live migration: number of Live Migrations that occurred

No more than one item of information can be requested per macro call. The desired information is output to the `<PREFIX><MACID>OUTP` field. The data type and length of the output field depend on the information requested, as shown in the table at the end of the operand description.

Macro format and description of operands

NSIINF
INFO=BIGPGSIZ / BS2ID / CONFNAMX / CPUIDIPL / (CPUID,8) / (CPUID,16) / EPOCH / HSIASF / HSIBASE / HSILINE / HSIVM / IPLDTTM / IPLMODE / NEWMSIZE / MEMSIZE / OSDVERS / OSIOID / PAGESIZE / SPSUFFIX / SYSNAME / MIGCOUNT ,SRVUNIT=STD / INITIAL / CURRENT ,MF=D / C / E / L [,PARAM=addr / (r)] ,PREFIX=N / p ,MACID=SII / macid

INFO=
Identifies the information to be output.
For contents and length of information, see table on page 704.

SRVUNIT =
Specifies the Server Unit whose data is to be displayed.
The explicit specification of this operand only makes sense if live migration has taken place and if the desired information depends on the Server Unit.

SRVUNIT = STD
The currently valid setting for the BS2000 session is to be used.
System-global default: INITIAL.

SRVUNIT = INITIAL
Server Unit on which IPL was performed (IPL server).

SRVUNIT = CURRENT
Server Unit on which the BS2000 session (possibly after live migration) is currently running.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29.
The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form or D form of the macro and additionally a MACID in the C form (see section “S-type macros” on page 29).

List of information identifiers and their meanings

Identifier	Length in bytes	Meanings of values output
BIGPGSIZ	4	Size of a “big page” in the memory (in bytes). This information is only meaningful on servers with x86 architecture. Otherwise it is rejected with the corresponding return code.
BS2ID		Information on the operating system being used. The information contains the following parts:
	8	<PREFIX> <MACID> BSNP: Program name (8 bytes, left-justified), e.g. 'BS2V190'
	10	<PREFIX> <MACID> BSVR: Version specification in DOD format, e.g. 'V19.0A00pp' pp is the PVLU information
	10	<PREFIX> <MACID> BSDT: Date of operating system generation in ISO4 format (YYYY-MM-DD)
	8	<PREFIX> <MACID> BSTM: Time of operating system generation in ISO4 format (HH:MM:SS)
CONFNAMX	21	Server type (model series) in the new expanded format, e.g. '7.500-␣S190-F␣␣␣␣␣␣␣␣' If the server type is not entered in the system 7.500-7.000␣␣␣␣␣␣␣␣␣␣␣ is output. The server designation is divided into the following four sections: Section 1: byte 0- 4: basic type Section 2: byte 6-10: model series Section 3: byte 12-15: model ID Section 4: byte 17-20: special model characteristics Bytes 5, 11 and 16 separate these sections and always contain the character '-' (hyphen). If there is no information on section 4, bytes 16-20 contain blanks. *Note* The old designation format of the server type can be taken from bytes 6 through 13 (`SINF INFO=CONFNAME`).
(CPUID,n)	8*n (n=8 or n=16)	Identifications of the CPUs. The nth element contains the identification of the nth CPU in hexadecimal form, as it is received from the hardware. If the nth CPU is not available, the nth element contains binary zero (8 times X'00'). The element number is then no longer identical to the CPU address. .
CPUIDIPL	8	Identification of the IPL CPU in hexadecimal form, as it is received from the hardware. See also (CPUID,n).
EPOCH	1	Epoch for the TOD register (see the “Introduction to System Administration” manual [10])

Identifier	Length in bytes	Meanings of values output
HSIASF	2	Provides additional information on the HSI. The following values may be output: AF: The operating system is running on a server with the option of extending the virtual address space. This permits optional access to a program space and a number of data spaces. This access is made possible by the use of an additional set of registers (access registers) (see section “Extended addressing with data spaces” on page 61). NA: No extended address space is available.
HSIBASE	6	CFCS3: HSI base type = CFCS3 (servers with /390 architecture). X86: HSI base type = X86 (servers with x86 architecture).
HSILINE	2	Provides additional information on the HSI. I: HSI base type CFCS3; CPU type = IX K: HSI base type X86; CPU type = KM UD: HSI base type not defined
HSIVM	2	Provides information as to whether a real or a virtual machine is present Possible values: V2 The operating system is running on a virtual machine under VM2000. NV The operating system is running on a real machine.
IPLDTTM	 10 8	Provides the following information: <PREFIX> <MACID> IPDA: Startup date of the current session in ISO4 format (YYYY-MM-DD) <PREFIX> <MACID> IPTM: Startup time of the current session in ISO4 format (HH:MM:SS)
IPLMODE	1	Provides information on the system startup mode. Possible mode: <PREFIX> <MACID> IPMD: dialog mode <PREFIX> <MACID> IPMA: automatic or fast mode
NEWMSIZE	4	Size of the (physical) main memory which can be used by the software (specified in megabytes)
MEMSIZE	4	Size of (physical) main memory available to software (in bytes). *Note* Out of preference, use NEWMSIZE, since MEMSIZE may not be sufficient for current memory sizes.
MIGCOUNT	4	Provides information on the number of live migrations which have taken place. 0 means that no live migrations have taken place in the session.

Identifier	Length in bytes	Meanings of values output
OSDVERS	10	Version of BS2000 in DOD format, e.g. 'V10.0A00pp' (pp is the PVLU information)
OSIOID	 8 10 10 8	Provides information on the I/O configuration of the operating system in use. The information contains the following parts: <PREFIX> <MACID> IONP: Program name (8 bytes, left-justified), e.g. 'OSIOV190' <PREFIX> <MACID> IOVR: Version specification in DOD format, e.g. 'V19.0A00pp' pp is the PVLU information <PREFIX> <MACID> IODT: Generation date of the I/O configuration in ISO4 format (YYYY-MM-DD) <PREFIX> <MACID> IOTM: Generation time of the I/O configuration in ISO4 format (HH:MM:SS)
PAGESIZE	4	hardware page size (specified in kilobytes)
SPSUFFIX	15	Provides information on the origin of the system parameters at system startup *NONE From <std-name> withou suffix *SYSTEM-NAME From <std-name>.<sys-name>, where <sys-name> corresponds to the system name (see the SYSNAME parameter) *VM-NAME From <std-name>.<vm-name>.<vm-name>, where <vm-name> corresponds to the VM name *IOCONF-ID-NAME From <std-name>.<ioconf-name>, where <ioconf-name> corresponds to the IOCONF program name *DIALOG The system parameters were entered either completely or partially via parameter files entered in the dialog or directly via the console. *UNKNOWN No information on name creation is available (only in the event of an error) <std-name> corresponds to SYSPAR.BS2.vvv, where vvv = BS2000 version without period; for example SYSPAR.BS2.190
SYSNAME	8	System name in the format <name 1..8>

Return information and error flags

The desired information is output to the `<PREFIX><MACID>OUTP` field.

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the NSIINF macro is transferred in the standard header (aaaa=Maincode, bb=Subcode1, cc=Subcode2).

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function executed successfully
X'01'	X'01'	X'0001'	No action: invalid specification for the INFO operand
X'02'	X'01'	X'0001'	No action: invalid specification in INFO=(CPUID,n)
X'06'	X'01'	X'0001'	No action: parameter cannot be used for current HSI (see BIGPGSIZ)

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Example

```
NSIINF   START
         PRINT NOGEN
NSIINF   AMODE ANY
         GPARMOD 31
         BALR   3,0
         USING  *,3
*
NSIOPLST NSIINF MF=D,INFO=BS2ID  -------------------------------------  (1)
NSIINF   CSECT
         LA    4,NSILIST
         USING NSIOPLST,4
         NSIINF MF=E,PARAM=NSILIST
         MVC   MESSNAME,=C'NP'
         MVC   MESSTXT(L'NSIIBSNP),NSIIBSNP
         BAL   7,OUTPUT                Call output routine ---------->
         MVC   MESSNAME,=C'VR'
         MVC   MESSTXT,NSIIBSVR
         BAL   7,OUTPUT                Call output routine ---------->
         MVC   MESSNAME,=C'DT'
         MVC   MESSTXT,NSIIBSDT
         BAL   7,OUTPUT                Call output routine ---------->
         MVC   MESSNAME,=C'TM'
         MVC   MESSTXT(L'NSIIBSTM),NSIIBSTM
         BAL   7,OUTPUT                Call output routine ---------->
END      TERM
*
*        Output routine
*
OUTPUT   WROUT MESSAGE,END,PARMOD=31
         MVI   MESSTXT,C' '            Clear MESSTXT for next output
         MVC   MESSTXT+1(L'MESSTXT-1),MESSTXT
         BR    7                       Return ->
*
****  Definitions ****
         DS    0H
NSILIST  NSIINF MF=L,INFO=BS2ID  -------------------------------------  (2)
MESSAGE  DC    Y(ENDMESS-MESSAGE)      Record length
         DS    CL2                     Reserved
         DC    X'01'                   Print feed control character
         DC    C'NSIIBS'
MESSNAME DC    CL2' '                  Field indicator
         DC    C' = '
MESSTXT  DC    CL10' '                 Contents
ENDMESS  EQU   *
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,nsiinf), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,nsiinf))
%  ASS6011 ASSEMBLY TIME: 368 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 85 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=nsiinf
%  BLS0523 ELEMENT 'NSIINF', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'NSIINF', VERSION ' ' OF '<date> <time>' LOADED
NSIIBSNP = I10BXS ———————————————————————————————————————————————————  (3)
NSIIBSVR = V19.0A00I1
NSIIBSDT = <date>
NSIIBSTM = <time>
```

(1) The DSECT for BS2ID (information on the operating system currently in use) is generated.

(2) The parameter list is supplied with the information.

(3) The following information regarding the current operating system is transferred:
I10BXS: Program name
V19.0A00I1: Version (V19.0A00) and PVLU information (I1)
<date>: Date on which the operating system was generated
<time>: Time at which the operating system was generated.

NSIOPT – Output system parameters

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Type S, MF format **3**: D/C/E/L form; see page 29

System parameters define specific options of an individual operating system.

System parameters are entered in the startup parameter file or the corresponding default values are used. System parameters can be modified individually during a system run. These modifications must also be saved in the startup parameter file if they are required for future system starts.

Macro description

The **NSIOPT** macro provides information on nonprivileged system parameters. When the caller has the TSOS privilege, the privileged systems parameters are also output.

All the system parameters are described in the appendix of the “Introduction to System Administration” [10].

Only one system parameter can be queried for each macro call. The INFO operand is used to select the system parameter. The desired information is transferred to an output field defined in the FIELD and LENG operands. The data type of the field depends on the system parameter being queried.

Macro format and description of operands

NSIOPT
[INFO='info',FIELD=addr,LENG=length] ,MF=D / C / E / L [,PARAM=addr / (r)] ,PREFIX=N / p ,MACID=SIO / macid

INFO=
Identifies the information to be output.

'info'
Identifier for the system parameter. The string “info” must be exactly 8 bytes long. If the name of the desired system parameter is less than 8 bytes long, it must be padded to this length with blanks.

FIELD=
Specifies the address of the field to which the information is to be output. The field length is determined by the value of the LENG operand. The information is entered left-justified.

addr
Symbolic address of the field.

LENG=
Specifies the length of the entry in the field specified in the FIELD operand. The length is given in the “Introduction to System Administration” manual [10]. If an incorrect value is specified here, no entry is made in the field specified with FIELD.

length
Length in bytes (see list), specified as a decimal digit.
For system parameters of type C, information can be output to a field which is too small, provided that only blanks are deleted from the contents of the system parameter.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29.
The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form or D form of the macro and additionally a MACID in the C form.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the NSIOPT macro is transferred in the standard header (aaaa=Maincode, bb=Subcode1, cc=Subcode2).

X'cc'	**X'bb'**	**X'aaaa'**	**Meaning**
X'00'	X'00'	X'0000'	Function executed successfully
X'01'	X'01'	X'0001'	No action: invalid specification for the INFO operand
X'02'	X'01'	X'0001'	No action: invalid specification for the FIELD operand or for the FIELD and LENG operands
X'03'	X'01'	X'0001'	No action: invalid specification for the LENG operand, e.g. no value specified for LENG
X'04'	X'01'	X'0001'	No action: the LENG operand does not match the length of the requested system parameter
X'05'	X'01'	X'0001'	No action: information on the requested system parameter is not available to nonprivileged users

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Example

```
  NSIOPT   START
  NSIOPT   AMODE ANY
           GPARMOD 31
           BALR   3,0
           USING  *,3
  *
  NSIOPLST NSIOPT MF=D ──────────────────────────────────────────────── (1)
1 *              NSIOPT, VERSION=101, DATE=900911                     101
1 NSIOPLST MFCHK MF=D,                                                101C
1                PREFIX=N,                                            101C
1                MACID=SIO,                                           101C
1                DMACID=SIO,                                          102C
1                DNAME=SIOPL,                                         101C
1                SUPPORT=(D,C,L,E),                                   101C
1                PARAM=,                                              101C
1                SVC=135                                              101
2 NSIOPLST DSECT ,
2                *,##### PREFIX=N, MACID=SIO #####
1 *                                                                   101
1          #INTF INTNAME=NSIOPT,REFTYPE=REQUEST,                      101C
1                INTCOMP=1                                            101
1 *                                                                   101
1 NSIORCNE EQU   0                       NO ERROR                     101
1 NSIORCII EQU   1                       INFO INVALID                 101
1 NSIORCFI EQU   2                       FIELD INVALID                101
1 NSIORCLI EQU   3                       LENG INVALID                 101
1 NSIORCLS EQU   4                       LENG TOO SHORT               101
1 *                                                                   101
1 NSIS0002     EQU  *                                                 101
1 NSIOFHDR FHDR  MF=(C,NSIO),EQUATES=NO                               101
2 NSIOFHDR DS    0A
2 NSIOFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 NSIOIFID DS    0A              0   INTERFACE IDENTIFIER
2 NSIOFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 NSIOFCT  DS    AL1             2   FUNCTION NUMBER
2 NSIOFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 NSIORET  DS    0A              4   GENERAL RETURN CODE
2 NSIOSRET DS    0AL2            4   SUB RETURN CODE
2 NSIOSR2  DS    AL1             4   SUB RETURN CODE 2
2 NSIOSR1  DS    AL1             5   SUB RETURN CODE 1
```

```
2 NSIOMRET DS    OAL2            6   MAIN RETURN CODE
2 NSIOMR2  DS    AL1             6   MAIN RETURN CODE 2
2 NSIOMR1  DS    AL1             7   MAIN RETURN CODE 1
2 NSIOFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1 NSIOINFO DS    CL8                     INFO                         101
1 NSIOFILD DS    A                       POINTER                      101
1 NSIOLENG DS    H                       LENGTH                       101
1 NSIE0002     EQU  *                                                 101
1 NSIO#        EQU  (NSIE0002-NSIS0002)                               101
           PRINT NOGEN
  NSIOPT   CSECT
           LA    4,NSILST
           USING NSIOPLST,4
           BAL   7,OUTPUT                Call output routine ----------->  (2)
           MVC   NSIOINFO,=C'SSMLGOF1' ---------------------------------  (3)
           MVC   NSIOLENG,=H'9'
           BAL   7,OUTPUT                Call output routine ----------->
           MVC   NSIOINFO,=C'ENCRYPT ' ---------------------------------  (4)
           MVC   NSIOLENG,=H'1'
           BAL   7,OUTPUT                Call output routine ----------->
  END      TERM
  *
  *        Output routine
  *
  OUTPUT   NSIOPT MF=E,PARAM=NSILST ------------------------------------  (5)
           MVC   PARNAME,NSIOINFO
           WROUT MESSAGE,END,PARMOD=31
2                *,@DCEO      999    921011   53531004
           MVI   OPTTXT,C' '             Clear OPTSTXT for next output
           MVC   OPTTXT+1(L'OPTTXT-1),OPTTXT
           BR    7                       Return ->
  *
  **** Definitions ****
           DS    0H
  NSILST   NSIOPT MF=L,INFO='BLKCTRL ',FIELD=OPTTXT,LENG=6 -------------  (6)
  MESSAGE  DC    Y(ENDMESS-MESSAGE)      Record length
           DS    CL2                     Reserved
           DC    X'01'                   Print feed control character
  PARNAME  DC    CL8' '                  Field indicator
           DC    C' = '
  OPTTXT   DC    CL10' '                 Contents
  ENDMESS  EQU   *
           END
                 =C'SSMLGOF1'
                 =C'ENCRYPT '
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,nsiopt), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,nsiopt))
%  ASS6011 ASSEMBLY TIME: 366 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 92 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=nsiopt
%  BLS0523 ELEMENT 'NSIOPT', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'NSIOPT', VERSION ' ' OF '<date> <time>' LOADED
BLKCTRL  = PAMKEY ─────────────────────────────────────────────────────  (7)
SSMLGOF1 = REQ-SPOOL
ENCRYPT  = Y
```

(1) Generation of DSECT for addressing the output information.

(2) First call of the output routine (5).

(3) Modification of the **NSIOPT** data area for the second call: The system parameter SSMLGOF1 should be output at a length of 9. The output routine (5) is then called for the second time.

(4) Modification of the **NSIOPT** data area for the third call: The system parameters ENCRYPT should be output at a length of 1. The output routine (5) is then called for the third time.

(5) Output routine.
The value of a system parameter is first determined using the macro **NSIOPT**. The determined value is then output to SYSOUT using **WROUT**.
The output routine is called three times in this example.

(6) Call to macro **NSIOPT** with MF=L for initialization of the data area. The file attribute BLKCTRL is to be output. The length of the information to be output is 6 bytes.

(7) Output of the determined system parameters:

- The system parameter BLKCTRL displays the value 'PAMKEY' as the default value for the BLKCTRL file attribute.
- The system parameter SSMLGOF1 displays the value 'REQ-SPOOL', i.e. spoolout jobs are always accepted.
- The system parameter ENCRYPT displays the value 'Y'. Passwords are entered in the file catalog in encrypted form.

OPCOM – Open communication

General

Application areas:	Intertask communication; see page 76 Communication; see page 163
Macro type:	Type O; see page 28

Macro description

The macro **OPCOM** allows the user to participate in intertask communication (ITC). The caller specifies a name, to be used as the ITC identification when sending or receiving messages. This name is entered in the ITC participants list, unless it is already being used by another ITC participant (see “Return information and error flags”, below).
The first participant to call the macro **OPCOM** implicitly causes ITC to be opened (a communication table is created).

Macro format and description of operands

OPCOM
name / (1)

name
Name which the caller wishes to use as an ITC name. This name may consist of up to 8 characters. Alphanumeric and special characters are permitted, except for:

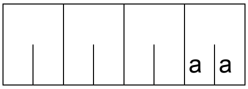
X'00' through X'3F'
X'41' through X'49'
X'51' through X'59'
X'62' through X'69'
X'70' through X'79'
X'80' through X'C0'
X'CA' through X'D0'
X'DA' through X'E1'
X'EA' through X'EF'
X'FA' through X'FE'

Character X'40' may only be specified as the last character of “name”.

(1)
Register R1 contains the address value of a field which contains the ITC name.
Field length = 8 bytes.

Return information and error flags

R15:



A return code relating to the execution of the OPCOM macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X' 00'	ITC participation opened
X' 04'	Operand error, ITC participation not opened
X' 08'	Name already in use, ITC participation not opened
X' 0C'	No system memory available to open ITC, or the internal maximum length for receive queues has been exceeded
X' 10'	ITC participation has already been enabled

OPSGEN – Control of S variable generation via MIP

General

Application area:	Message system; see page 161
Macro type:	Type S; see page 29

Macro description

The **OPSGEN** macro controls the starting and termination of S variable generation by the message system and transfers the variable name under which the variable created is to be stored.

Macro format and description of operands

OPSGEN
[SIGNAL=*START / *STOP]
,OUTPUT=*SYSOUT / *NONE
,MODE=*REPLACE / *EXTEND
[,MSGVAR=addr / (r)]
[,MSGVARL=length / addr / (r)]
[,MF=D / C / E / L / M]
[,PARAM=addr / (r)]
,PREFIX=N / p
,MACID=MHG / macid

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “macro type” and are included in the macro format.

SIGNAL=
Gives the current task the signal to start or terminate generation of structured message output to S variables. The output contains the message text, message code, inserts and reply area.

> ***START**
> If this signal is given, MIP starts to generate structured message output and to store it in S variables.

***STOP**
Generation of structured message output is terminated.

OUTPUT=
Determines whether or not messages from the program should be output to SYSOUT.

***SYSOUT**
Messages are output to SYSOUT.

***NONE**
Messages are discarded and not output.

MSGVAR=
Shows the address of an area containing the name of the required variable.
May only be specified if MF=M.

addr
Symbolic address of an area which again contains the address of the variable name.

(r)
Register containing the address of the area.

MSGVARL=
Specifies the length (max. 255) of the area addressed in MSGVAR.

length
Length of field to be specified in bytes.

addr
Symbolic address of a field 2 bytes long containing the length. This value may only be specified in conjunction with MF=M.

(r)
Register containing the address of the 2 byte field.

MODE=
Specifies whether the existing contents of the S variable should be overwritten or whether the newly generated messages should be added to the old contents.

***REPLACE**
S variable contents are deleted prior to description with the newly generated messages.

***EXTEND**
The newly generated messages are added to the existing S variable contents.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the OPSGEN macro is transferred in the standard header (aaaa=maincode,bb=subcode1,cc=subcode2).

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function executed successfully
X'00'	X'01'	X'0002'	Error in the parameter list
X'00'	X'01'	X'FFFF'	Function not known
X'00'	X'02'	X'FFFF'	Incorrect specification of UNIT/FUNCTION in standard header
X'00'	X'03'	X'FFFF'	Incorrect specification of VERSION in standard header
X'00'	X'20'	X'0003'	Problems with variable generation
X'00'	X'20'	X'0004'	Error in execution of function
X'00'	X'40'	X'0001'	SIGNAL=*STOP operand specified without prior switching on of variable generation with SIGNAL=*START

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

PASS – Wait one second

General

Application area:	Starting, interrupting and terminating; see page 72
Macro type:	Type O; see page 28

Macro description

The **PASS** macro causes the task to wait one second.

Macro format

PASS

Example

```
PASS     START
         PRINT NOGEN
         BALR  3,0
         USING *,3
         GEPRT ,CPU                  REMAINING PROGRAM TIME
         GDATE TOD=CLOCK             TIME OF DAY
         PASS                        WAIT ONE SECOND
         GDATE TOD=CLOCK1            TIME OF DAY
         GEPRT ,CPU1                 REMAINING PROGRAM TIME
DTH1     TERM
CLOCK    DS    CL8
         DC    C' '
CLOCK1   DS    CL8
         DC    C' '
CPU      DS    CL6
         DC    C' '
CPU1     DS    CL6
         DC    C' '
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,pass), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,pass)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 322 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 78 MSEC
//end
%  ASS6012 END OF ASSEMBH
/load-executable-program library=macexmp.lib,element-or-symbol=pass, -
/     test-options=*aid
%  BLS0523 ELEMENT 'PASS', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'PASS', VERSION ' ' OF '<date> <time>' LOADED
/%on %term<%d clock,clock1,cpu,cpu1>
/%r
*** TID: 005000D8 *** TSN: 2QSE *********************************************
**
SRC_REF:    54 SOURCE: PASS  PROC: PASS  ************************************
**
CLOCK          = |13:12:36| ------------------------------------------  (1)
CLOCK1         = |13:12:37|
CPU            = |022900| --------------------------------------------  (2)
CPU1           = |022900|
```

(1) Output of the time, before and after running the macro **PASS**. One second has passed.

(2) Output of the CPU time before and after running the macro **PASS**. No CPU time was used.

PINF – Output global program information

General

Application area:	Linking and loading; see page 47
Macro type:	Type S, MF format **2**: standard/C/D/L/E/M form; see page 29

See also the “BLSSERV” manual [4] for information in the dynamic binder loader DBL.

Macro description

The **PINF** macro provides the user with information on programs that were loaded by means of the LOAD-EXECUTABLE-PROGRAM or START-EXECUTABLE-PROGRAM (or LOAD-PROGRAM or START-PROGRAM) command.

The following information can be requested:

- the internal program name (SELECT=INTNAME),
- the internal program version (SELECT=INTVERS),
- the creation date of the program (SELECT=INTDATE),
- the copyright name of the program (SELECT=COPRIGHT),
- the name of the library in which the program is stored (SELECT=FILENAME),
- the element name of the program in the library (SELECT=ELEMNAME),
- the element version (SELECT=ELEMVERS),
- the element type (SELECT=ELEMTYPE),
- the name that was specified in the load call in the START-PROGRAM or LOAD-PROGRAM command (SELECT=SPECNAME),
- an indicator that shows whether the program was loaded by the static loader ELDE or was the first module of a load unit and was loaded by DBL (SELECT=LOADTYPE).

All information is displayed in the output field in hexadecimal form.

Macro format and description of operands

PINF
SELECT=INTNAME / INTVERS / INTDATE / COPRIGHT / FILENAME / ELEMNAME / ELEMVERS / ELEMTYPE / SPECNAME / LOADTYPE ,VERSION=001 / 002 ,ADDR=addr / (r) [,LEN=integer] ,MF=S / C / D / E / L / M [,PARAM=addr / (r)] ,PREFIX=P / p ,MACID=BPI / macid

The operands are described in alphabetical order below.

ADDR=addr
Specifies the symbolic address of a field to which DBL is to transfer the information.

> **(r)**
> Register containing the address value “addr”. May be specified only in conjunction with MF=M.

LEN=
Specifies the length in bytes of the output field specified in ADDR.
If this operand is not specified, the minimum length (4 bytes) is used.

> **integer**
> The minimum length is 4 bytes. If several items of information are specified in SELECT, the total length of the output field must be the sum of the lengths of the information items (see the SELECT operand).

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM, PREFIX and MACID, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

SELECT=

Specifies the type of information to be output. The symbolic names of the desired items of information must be specified here. Up to 8 items of information may be requested simultaneously. The following tables list the items of information that can be requested, giving their symbolic names, the length of the information and a description.

Meaning of the output with VERSION=001

The letters in parentheses in the table have the following meanings:

(D): The program was loaded by the dynamic binder loader DBL. The information refers to the first module (LLM or object module) of the load unit to be loaded.

(S): The program was loaded by the static loader ELDE (load module, C-type library element) or it is a PAM-LLM that was loaded by the dynamic binding loader DBL.

Symb. name	Length (bytes)	Description
INTNAME	41	Internal program name: (D): – internal name of the LLM or the first CSECT name of the object module, if it was loaded from the EAM object module file (S): – Internal name of the load module or PAM-LLM
INTVERS	24	interne Programmversion: (D): – Internal version of the LLM (S): – Internal version of the program or PAM-LLM
INTDATE	10	Creation date of the program in yyyy-mm-dd format yyyy=year, mm=month, dd=day (D): For LLM: creation date of LLM For object module: execution date of the R-type element or of the element from the EAM object module file (S): Creation date of load module or PAM-LLM
COPRIGHT	64	Copyright name of the program (D): Copyright name of the LLM (S): Copyright name of the load module or PAM-LLM
FILENAME	54	(D): Name of the library (program library or OML); if the program was loaded from the EAM object module file, the field contains blanks (S): Name of the cataloged file (of the load module or PAM-LLM) or the name of the library containing the C-type element (load module)

Symb. name	Length (bytes)	Description
ELEMNAME	64	(D): For LLM: the L-type element name. For object modules, one of the following: – the R-type element name in the program library – the element name in an OML or – the first CSECT name of the object module if it was loaded from the EAM object module file. (S): Name of the C-type element in a program library.
ELEMVERS	24	(D): For LLM: the version of the L-type element. For object module: the version of the R-type element in the program library (S): The version of the C-type element in the program library.
ELEMTYPE	8	Contains a letter identifying the element type and 7 blanks. (D): L = LLM or R = object module (S): C = load module
SPECNAME	64	Contains the name that was specified in the load call in the START-PROGRAM or LOAD-PROGRAM command: (D): The specified symbol or the string “EAM OMF” for the EAM object module library (S): The name of the cataloged file (of the load module or PAM-LLM) or the name of the C-type element in a program library.
LOADTYPE	1	Indicator for the type of the loaded program: = 0 : (S) – The program was loaded by the static loader ELDE or it is a PAM-LLM. ≠ 0 : (D) – The program was loaded by DBL and the information refers to the first module of the load unit.

Meaning of the output with VERSION=002

The letters in parentheses in the table have the following meanings:

(D): The program was loaded by the dynamic binder loader DBL. The information refers to the first module (LLM or object module) of the load unit to be loaded or to a PAM-LLM.

(S): The program was loaded by the static loader ELDE (load module, C-type library element).

Symb. name	Length (bytes)	Description
INTNAME	41	Internal program name: (D): – internal name of the LLM or PAM-LLM or the first CSECT name of the object module, if it was loaded from the EAM object module file (S): – Internal name of the load module
INTVERS	24	interne Programmversion: (D): – Internal version of the LLM or PAM-LLM (S): – Internal program version
INTDATE	10	Creation date of the program in yyyy-mm-dd format yyyy=year, mm=month, dd=day (D): For LLM/PAM-LLM: creation date of LLM/PAM-LLM For object module: execution date of the R-type element or of the element from the EAM object module file (S): Creation date of load module
COPRIGHT	64	Copyright name of the program (D): Copyright name of the LLM or PAM-LLM (S): Copyright name of the load module
FILENAME	54	(D): Name of the library (program library or OML) or filename of the PAM-LLM; if the program was loaded from the EAM object module file, the field contains blanks (S): Name of the cataloged file (of the load module) or the name of the library containing the C-type element (load module)
ELEMNAME	64	(D): For LLM: the L-type element name. For object modules, one of the following: – the R-type element name in the program library – the element name in an OML or – the first CSECT name of the object module if it was loaded from the EAM object module file. (S): Name of the C-type element in a program library.
ELEMVERS	24	(D): For LLM: the version of the L-type element. For object module: the version of the R-type element in the program library (S): The version of the C-type element in the program library.

Symb. name	Length (bytes)	Description
ELEMTYPE	8	Contains a letter identifying the element type and 7 blanks. (D): L = LLM or R = object module (S): C = load module
SPECNAME	64	Contains the name that was specified in the load call in the START-PROGRAM or LOAD-PROGRAM command: (D): The specified symbol or the string “EAM OMF” for the EAM object module library or the filename of the PAM-LLM (S): The name of the cataloged file (of the load module) or the name of the C-type element in a program library.
LOADTYPE	1	Indicator for the type of the loaded program: = 0 : (S) – The program was loaded by the static loader ELDE. ≠ 0 : (D) – The program was loaded by DBL and the information refers to the first module of the load unit.

VERSION=
Defines the required macro version and thus the layout of the output.

001
Version 1 of the **PINF** macro is used. In this case no distinction is made between load modules and PAM-LLMs in the output.

002
Version 2 of the **PINF** macro is used. This specification is supported as of BLSSERV V2.5.

The information output for PAM-LLMs is different from the information for load modules.

Notes on the macro call

A macro in the form MF=M is normally used to modify the parameter list dynamically.

If, however, the parameter list is modified directly with MF=D by means of symbolic names, the user should bear in mind the following points for the macro processor:

- The entries must be made sequentially from the first position.
- Entries that follow the entries currently available are ignored.
- A null entry amongst the entries currently available is ignored, but is nevertheless treated as a valid entry.

Return information and error flags

The requested information is transferred to the field specified in the ADDR operand.

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the PINF macro is transferred in the standard header (cc=Subcode2,bb=Subcode1,aaaa=Maincode)

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally
X'00'	X'40'	X'0001'	The desired information is not yet defined (warning) or is no longer supported.
X'00'	X'01'	X'0010'	The output field is too small
X'00'	X'01'	X'0020'	An unknown symbolic name was specified in SELECT
X'00'	X'01'	X'0070'	The memory area for the output field is not assigned
X'00'	X'20'	X'0090'	Internal error
X'00'	X'01'	X'0100'	The SELECT operand is missing
X'00'	X'01'	X'0110'	More than 8 items of information were specified for SELECT
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported
X'00'	X'02'	X'FFFF'	The function is not available
X'00'	X'03'	X'FFFF'	The version of the interface is not supported

Example

The PINFBSP program calls the **PINF** macro and outputs the following information:

- the internal program name
- the library name
- the name of the library element
- the element type
- the element version
- the name specified in the load call.

```
PINFBSP  CSECT
PINFBSP  AMODE ANY
PINFBSP  RMODE ANY
         BASR   10,0
         USING  *,10
         GPARMOD 31
         PRINT  GEN
         EJECT
*
***** PINF SELECT = INTNAME *****
*
INTNAM   MVC    MTEXT(24),=CL24'PINF SEL=INTNAME'
         MVC    MEXPECT(4),=XL4'00000000'
         PINF   SELECT=INTNAME,ADDR=ZONE,LEN=41 ────────────────────────   (1)
         L      15,4(1) ────────────────────────────────────────────────   (2)
         CL     15,MEXPECT
         BE     WINTNAM ────────────────────────────────────────────────   (3)
         BAS    12,MEREXP ──────────────────────────────────────────────   (4)
         B      FILNAM ─────────────────────────────────────────────────   (5)
WINTNAM  MVC    MAREA,MTEXT ────────────────────────────────────────────   (6)
         MVC    ZAREA(41),ZONE
         BAS    12,WRINFO
*
***** PINF SELECT = FILENAME *****
*
FILNAM   MVC    MTEXT(24),=CL24'PINF SEL=FILENAME'
         MVC    MEXPECT(4),=XL4'00000000'
         PINF   SELECT=FILENAME,ADDR=ZONE,LEN=54 ───────────────────────   (7)
         L      15,4(1)
         CL     15,MEXPECT
         BE     WFILNAM
         BAS    12,MEREXP
         B      ELMT
WFILNAM  MVC    MAREA,MTEXT ────────────────────────────────────────────   (8)
         MVC    ZAREA(54),ZONE
         BAS    12,WRINFO
*
```

```
***** PINF SELECT = ELEMNAME, ELEMVERS, ELEMTYPE *****
*
ELMT     MVC    MTEXT(24),=CL24'PINF SEL=ELEMxxxx'
         MVC    MEXPECT(4),=XL4'00000000'
         PINF   SELECT=(ELEMNAME,ELEMVERS,ELEMTYPE),ADDR=ZONE,LEN=96 ——  (9)
         L      15,4(1)
         CL     15,MEXPECT
         BE     WELMT
         BAS    12,MEREXP
         B      SPNAM
WELMT    MVC    MAREA(24),=CL24'PINF SEL=ELEMNAME' ———————————————————— (10)
         MVC    ZAREA(64),ZONE
         BAS    12,WRINFO
*
         MVC    MAREA(24),=CL24'PINF SEL=ELEMVERS' ———————————————————— (11)
         MVC    ZAREA(24),ZONE+64
         BAS    12,WRINFO
*
         MVC    MAREA(24),=CL24'PINF SEL=ELEMTYPE' ———————————————————— (12)
         MVC    ZAREA(8),ZONE+88
         BAS    12,WRINFO
*
***** PINF SELECT = SPECNAME *****
*
SPNAM    MVC    MTEXT(24),=CL24'PINF SEL=SPECNAME'
         MVC    MEXPECT(4),=XL4'00000000'
         PINF   SELECT=SPECNAME,ADDR=ZONE,LEN=64 —————————————————————— (13)
         L      15,4(1)
         CL     15,MEXPECT
         BE     WSPNAM
         BAS    12,MEREXP
         B      MTERM
WSPNAM   MVC    MAREA,MTEXT ——————————————————————————————————————————— (14)
         MVC    ZAREA(64),ZONE
         BAS    12,WRINFO
*
MTERM    TERM —————————————————————————————————————————————————————————— (15)
MTERMD   TERM DUMP=Y ——————————————————————————————————————————————————— (16)
*
****  Definitions  ****
MEREXP   DS     0H
         MVC    MPACK(4),MEXPECT —————————————————————————————————————— (17)
         UNPK   MEXP+1(9),MPACK(5)
         NC     MEXP+1(8),=X'0F0F0F0F0F0F0F0F'
         TR     MEXP+1(8),MTAB
         MVI    MEXP,C'('
         MVC    MEXP+9(10),MCEXP
*
MEREAL   ST     15,MPACK ---------------------------------------------- (18)
         UNPK   MRS(9),MPACK(5)
         NC     MRS(8),=X'0F0F0F0F0F0F0F0F'
         TR     MRS(8),MTAB
         MVI    MRS+8,MBLANK
*
MEWROUT  DS     0H
         WROUT  MMSG,MTERMD -------------------------------------------- (19)
         MVI    MTEXT,MBLANK ------------------------------------------- (20)
         MVC    MTEXT+1(51),MTEXT
         BR     12
*
WRINFO   DS     0H
         WROUT  AREA,MTERMD -------------------------------------------- (21)
         MVI    ZAREA,MBLANK
         MVC    ZAREA+1(95),ZAREA
         BR     12
*
**** DATA DEFINITION ****
*
AREA     DC     Y(LAREA)
         DC     CL3' '
MAREA    DC     CL24' '
ZAREA    DC     CL96' '
LAREA    EQU    *-AREA
MMSG     DC     Y(MMSGE)
         DC     C'    ==ERROR==     '
MTEXT    DC     CL24' '
MRS      DC     CL9' '
MEXP     DC     CL19' '
MMSGE    EQU    *-MMSG
MBLANK   EQU    X'40'
MCEXP    DC     C' EXPECTED)'
MPACK    DS     F
         DC     X'00'
MTAB     DC     C'0123456789ABCDEF'
MEXPECT  DC     F'0'
ZONE     DS     CL96
         END
```

(1) The **PINF** macro is called. The internal program name is requested; this name has a maximum length of 41 bytes and is to be stored in the `ZONE` field.

(2) The macro return code, which is stored in the second word of the standard header, is loaded into register R15. Then the actual return code is compared with the expected return code X'00000000'.

(3) Once the macro has been executed successfully, the program goes on to output the information returned by **PINF**.

(4) In the event of unsuccessful macro execution, the program branches to the error handling routine.

(5) Once error handling is complete, the program continues with the next **PINF** call.

(6) The internal program name written to the ZONE field by the **PINF** macro is output.

(7) The **PINF** macro is called again. The name of the program library is requested; this name has a maximum length of 54 bytes and is to be stored in the `ZONE` field.

(8) The library name is output.

(9) The **PINF** macro is called again. The following items of information are requested:
– the name of the element in the program library,
– the element version,
– the element type.
A total of 96 bytes must be reserved for these three items of information.

(10) The element name is output. It begins at the start of the `ZONE` field and has a length of 64 bytes.

(11) The element version is output. It starts at the symbolic address ZONE+64 and has a length of 24 bytes.

(12) The element type is output. It starts at the symbolic address ZONE+88 and has a length of 8 bytes.

(13) The **PINF** macro is called again. The symbol name specified in the load call (e.g. in START-EXECUTABLE-PROGRAM) is requested. The symbol name has a maximum length of 64 bytes and is to be stored in the `ZONE` field.

(14) The symbol name specified in the load call is output.

(15) The program is terminated normally.

(16) If errors occurred in the **WROUT** macro, the program is terminated with a dump.

(17) The expected macro return code is edited as a string to be output at a later time.

(18) The actual macro return code is edited as a string to be output at a later time.

(19) An error message is issued if the **PINF** macro returned a return code ≠ X'00000000'. In this case, the following message is issued:

```
==ERROR==     PINF SEL=zzzzzzzz      xxxxxxxx (00000000 EXPECTED)
```

where:
`zzzzzzzz` is the selected information
`xxxxxxxx` is the value of the actual macro return code

(20) The fields MTEXT, MRS and MEXP are assigned blanks (X'40') and the program is continued.

(21) The requested program information is output. The ZAREA field is then assigned blanks (X'40') and the program is continued.

Runtime log

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,pinfbsp), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,pinfbsp))
%  ASS6011 ASSEMBLY TIME: 381 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 143 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=pinfbsp
%  BLS0523 ELEMENT 'PINFBSP', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'PINFBSP', VERSION ' ' OF '<date> <time>' LOADED
PINF SEL=INTNAME        PINFBSP
PINF SEL=FILENAME       :2OSG:$QM212.MACEXMP.LIB
PINF SEL=ELEMNAME       PINFBSP
PINF SEL=ELEMVERS       ~
PINF SEL=ELEMTYPE       L
PINF SEL=SPECNAME       PINFBSP
```

(22) The requested information is output to SYSOUT.
The internal program name is PINFBSP. The module was fetched from the program library MACEXMP.LIB, where it is stored as an LLM with the name PINFBSP. The name PINFBSP was specified in the load call (in START-EXECUTABLE-PROGRAM).

POSSIG – Post signal request

General

Application area:	Eventing; see page 94
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

In the event of macro chaining, all the macros chained must make use of the same interface. If the 24-bit interface is used, a 4-byte field is generated for the post code. If the 31-bit interface is used, the post code may have a length of either 4 or 8 bytes.

Macro description

This macro is used to indicate to the event item that an event has occurred. The event item must previously have been assigned to the task of the calling program (by calling the **ENAEI** macro).
The program which issued the **POSSIG** macro is always continued. It can also specify a contingency process, which is initiated as soon as **POSSIG** satisfies a “SOLicit SIGnal request” (SOLSIG) or as soon as a prescribed period of time has elapsed. This contingency process provides information on whether or not the signal was used within this period of time.
The contingency process must already have been defined (by **ENACO**).

Macro format and description of operands

```
POSSIG
--------------------------------------------------------------------------

{ { EINAME=name                                      }         { LOCAL      } }
{ {                                                  },SCOPE=  { GROUP      } }
{ { EINAMAD= { addr } [,EINAMLN=length]              }         { USER_GROUP } }
{ {          { (r)  }                                }         { GLOBAL     } }
{                                                                             }
{ EIID= { addr }                                                              }
{       { (r)  }                                                              }

[, { SPOSTAD= { addr } } ],SPOSTL=1/ 2
   {          { (r)  } }
   { SPOSTR=r          }
```

POSSIG (cont.)
[,LIFETIM=sec / (r)] ,CONTINU=NO / YES / SOLSIG [,COID=addr / (r)] [,COMAD=addr / (r)] [,PARMOD=24 / 31] [,MF=L / (E,..)]

EINAME=name
Specifies the name of the event item to which the event is to be signaled. The event item must previously have been defined by the **ENAEI** macro. The name of the event item is unique only if SCOPE is also specified.

EINAMAD=
Specifies the address of the name of the event item. This entry is unique only if SCOPE is also specified.

addr
Symbolic address of the field containing the name.

(r)
Register containing the address.

EINAMLN=
Specifies the length in bytes of the event item name. If the operand is missing, the length attribute of the EINAMAD operand is assumed if EINAMAD=addr is specified;
if EINAMAD=(r), the maximum length (54 bytes) is assumed.

length
Length of the event item name.

SCOPE=
Specifies the scope of the event item (i.e. participants authorized to use it).

LOCAL
The use of the event item is limited to the calling task.

GROUP
All the tasks with the same user ID as the calling task are participants.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.
The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP; the program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

EIID=
Specifies the ID of the event item. The ID is supplied to the user by the **ENAEI** macro call. If the ID is used instead of the name to identify the event item, it speeds up processing. The ID is unique.

addr
Symbolic address of a 4-byte field containing the event item ID.

(r)
Register containing the address of the field.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

SPOSTAD=
Specifies the field containing the post code information to be transferred to the corresponding program (which issues the **SOLSIG** macro) (see section “Eventing” on page 94). The post code has a length of either 4 or 8 bytes.
It is also transferred to the contingency process, if present (COID operand). Post code X'00000000' is not transferred.
Processing is speeded up if SPOSTR, which offers the same function, is used instead of SPOSTAD.

addr
Symbolic address of the field containing the post code information.
Length = 4 or 8 bytes.

(r)
Register containing the address value “addr”.

SPOSTR=
Specifies a register containing the post code information to be transferred to the corresponding program (which issues the **SOLSIG** macro).
If the 8-byte post code (= 2 words) is used, the register following the specified register (in number) must contain the 2nd word of the post code. Post code X'00000000' is not transferred.

(r)
Register containing the post code. Registers R0 and R1 should not be used.

SPOSTL=
Gives the length of the post code in words. If the 24-bit interface (PARMOD=24) is used, only SPOSTL=1 is permitted.

<u>1</u>
The post code is 1 word (4 bytes) long.

2
The post code is 2 words long.

LIFETIM=
Specifies the time for which the signal is to be used by a corresponding SOLSIG request. The event information code informs the contingency process, if any, whether or not the signal was used within this period of time.

sec
Time in seconds. $1 \leq \text{sec} \leq 43200$
The processing accuracy for this operation is +10 seconds.
Default value: 600 sec.

(r)
Register containing the time specification in seconds.

CONTINU=
Permits **POSSIG** macro chaining with further **POSSIG** macros or with a **SOLSIG** macro. All macros chained must make use of the same interface.

NO
No further **POSSIG** or **SOLSIG** macro immediately follows the **POSSIG** macro.

YES
A further **POSSIG** macro immediately follows a **POSSIG** macro.

SOLSIG
A **SOLSIG** macro immediately follows the **POSSIG** macro.

Operands used to specify a contingency process

COID=
Specifies the ID of the contingency process. The ID is supplied to the user by the **ENACO** macro.

addr
Symbolic address of the field containing the ID.

(r)
Register containing the address value “addr”.

COMAD=
Specifies a contingency message. This contingency message is passed to the contingency process (register R1). A message issued here replaces any message that might have been issued when the contingency was defined (see section “Contingency processes” on page 110).

addr
Symbolic address of a word containing a contingency message.

(r)
Register containing the address.

Notes on the macro call

- The POSSIG queue of an event item cannot accept an unlimited number of requests. In order to protect the system, the number of POSSIG requests in the queue of an event item is limited to a machine-dependent maximum value.
- If a program (package) has defined a contingency process that is written in SPL, register 12 must contain the address of the SPL program manager for all **ENACO**, **SOLSIG** and **POSSIG** calls.

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:

b	b					a	a

A structured return code (aa = primary return code, bb = secondary return code) relating to the execution of the POSSIG macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'00'	X'00'	Function executed: the signal was successfully sent
X'0C'	X'04'	No action: the event item established by the system was not assigned to the task of the calling program
X'10'	X'04'	No action: invalid operands were specified
X'14'	X'04'	No action: invalid name or ID. An event item with the specified name does not exist
X'18'	X'04'	No action: the maximum number (400) of contingency processes permitted per basic task has been exceeded
X'24'	X'04'	No action: invalid contingency ID. No contingency process with this ID exists
X'28'	X'04'	No action: the maximum number of requests in the POSSIG queue was reached

The following applies to macro chaining (CONTINU operand):

aa = X'00' All macro calls in the chain were completed.

aa = X'04' A macro in the chain was not completed (all preceding macros in the chain were completed). The chain is aborted at this point.
The error cause is contained in the corresponding secondary indicator.

Information on post code usage is supplied on page 96.

The various meanings of the event information code values for contingency processes are described in table 8 on page 117.

Example

```
  POSSIG   START
           PRINT NOGEN
  POSSIG   AMODE ANY
           GPARMOD 31
1                *,MACRO: GPARMOD, VERSION: VER121
           BALR  5,0
           USING *,5
           ENAEI EINAME=EVENT,SCOPE=GROUP,EIIDRET=KKEV ────────────────  (1)
           POSSIG EIID=KKEV ───────────────────────────────────────────  (2)
           ST    15,RCFIELD1
           ENACO CONAME=CONT,COADAD=COANFAD,COIDRET=KKCO ──────────────  (3)
           POSSIG EIID=KKEV,COID=KKCO,LIFETIM=60 ──────────────────────  (4)
           ST    15,RCFIELD2
  LOOP     CLI   SWITCH,'0'
           BE    LOOP
           ST    2,ACKNO ──────────────────────────────────────────────  (5)
           DISCO COID=KKCO
           ST    15,RCFIELD3
           DISEI EIID=KKEV
           ST    15,RCFIELD4
  DTH1     TERM
  COANF    BALR  6,0
           USING *,6
           CONTXT STACKR=(2),OWNR=(2),FUNCT=WRITE ─────────────────────  (6)
           MVI   SWITCH,'1'
           RETCO
  KKEV     DS    F
  KKCO     DS    F
  COANFAD  DC    A(COANF)
  SWITCH   DC    C'0'
  *
  ACKNO    DS    F
  RCFIELD1 DS    F
  RCFIELD2 DS    F
  RCFIELD3 DS    F
  RCFIELD4 DS    F
           END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,possig), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,possig)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 509 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 84 MSEC
//end
%  ASS6012 END OF ASSEMBH
/load-executable-program library=macexmp.lib,element-or-symbol=possig, -
/     test-options=*aid
%  BLS0523 ELEMENT 'POSSIG', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'POSSIG', VERSION ' ' OF '<date> <time>' LOADED
/%in dth1;%r
STOPPED AT LABEL: DTH1 , SRC_REF: 158, SOURCE: POSSIG , PROC: POSSIG
/%d %@(rcfield1) -> %x
*** TID: 005000D8 *** TSN: 2QSE *********************************************
CURRENT PC: 000000A6    CSECT: POSSIG  **************************************
V'000000FC' = POSSIG   + #'000000FC' ---------------------------------- (7)
000000FC (000000FC) 00000000                               ....
/%d %@(rcfield2) -> %x
V'00000100' = POSSIG   + #'00000100' ---------------------------------- (8)
00000100 (00000100) 00000000                               ....
/%d %@(quitt) -> %x
V'000000F8' = POSSIG   + #'000000F8' ---------------------------------- (9)
000000F8 (000000F8) 00000004                               ....
/%d %@(rcfield3) -> %x, %@(rcfield4) -> %x ---------------------------- (10)
V'00000104' = POSSIG   + #'00000104'
00000104 (00000104) 04000000                               ....
V'00000108' = POSSIG   + #'00000108'
00000108 (00000108) 04000000                               ....
/%r
```

(1) The event item EVENT is defined. KKEV is the address of the ID.

(2) The **POSSIG** macro is used to post a signal to the event item. A contingency process as acknowledgment is not specified. As no other participant requests the signal by means of **SOLSIG**, the signal remains in the POSSIG event queue until the waiting time has expired (here 600 seconds, default value).

(3) The COANF contingency process is defined by the ID address KKCO (see the **ENACO** macro).

(4) A second signal is posted to the event item. The contingency process (ID address KKCO) is to be started after 60 seconds unless a previous **SOLSIG** macro has already initiated the start. After this **POSSIG** call the program idles in a wait loop.

(5) The basic process stores register 2 (event information code) under ACKNO, closes the contingency definition (**DISCO**) and disables eventing (**DISEI**).

(6) The COANF contingency process is started. The event information code in register 2 specifies that no **SOLSIG** macro arrived. By means of the **CONTXT** macro it is transferred from register 2 of the contingency process to register 2 of the basic process (see the **CONTXT** macro). After this the loop variable is changed to open the idle loop and the contingency process is terminated by means of the **RETCO** macro.

(7) Return switch after the first **POSSIG** macro (see (2)): posting of signal was successful.

(8) Return switch after the second **POSSIG** macro (see (4)): posting of signal was successful.

(9) Event information code of contingency process: the expected event did not occur within the waiting time. Neither contingency message nor post code exists.

(10) Return switch after the **DISCO** and **DISEI** macros: the contingency definition and the event item were disabled.

For further **examples**, see the sections “Eventing” (page 106) and “Contingency processes” (page 118)

RDATA – Read record from SYSDTA

General

Application areas:	Input/output of files and records; see page 156 Data terminal communication; see page 160
Macro type:	Type S, MF format **1**: 24-bit interface: standard/E/L form 31-bit interface: standard/E/L/C/D form; see page 29

This macro description applies to TIAM V13.2A

The following applies when using the 31-bit interface:

– Neither symbolic names nor equates are generated for the standard header when MF=C/D is used. In the event of the operand list being supplied dynamically, the initialization values for the standard header should be taken from an operand list generated with MF=L.
A user dump is produced if the UNIT field is supplied with an incorrect value.

– No return code is transferred in the standard header.

The **CUPAB** macro generates a DSECT of the **RDATA** operand list for the 24-bit interface format.

Macro description

RDATA enables the next data record to be read from SYSDTA.
SYSDTA can be assigned to a PLAM library element, a cataloged SAM or ISAM file or - typically - the user's terminal.
The record (or, in the case of the terminal, the message) is placed in an area of the user program as a variable-length record.

The keyboard is locked for the 8160, 9749 and 975x Data Display Terminals after an input with **RDATA**. This means that no further inputs are possible before the next output; only short codes are permitted.

If a “BREAK” is detected during the read operation, the program counter is reset to the start of the macro expansion, with the result that the macro is repeated after the interrupt has been processed.

On execution of the macro (in the case of format 1), the specified operands are stored in an operand table and the start address of this table is loaded into register R1. In the case of format 2, the table specified in the user program is used.

Macro format 1 and description of operands

```
RDATA
--------------------------------------------------------------------------
record,error[,length][,edit][,A]
,KEYOUT=N / Y
,KEYPOS=N / Y
,KEYLEN=N / Y
  ⎧                                                                 ⎫
  ⎪ ,MODE=COMP ,ITRSUP=⎧ NO  ⎫,ILINEND=⎧ NO  ⎫                       ⎪
  ⎪                    ⎩ YES ⎭         ⎩ YES ⎭                       ⎪
  ⎪                                                                 ⎪
  ⎪            ,ILCASE=⎧ NO  ⎫,IHDR=⎧ NO  ⎫                          ⎪
  ⎪                    ⎩ YES ⎭      ⎩ YES ⎭                          ⎪
  ⎪                                                                 ⎪
  ⎪            ,IGETBS=⎧ NO  ⎫                                       ⎪
  ⎪                    ⎩ YES ⎭                                       ⎪
[ ⎨                                                                 ⎬]
  ⎪ ,MODE=LINE ,ILCASE=⎧ NO  ⎫,IGETBS=⎧ NO  ⎫                        ⎪
  ⎪                    ⎩ YES ⎭        ⎩ YES ⎭                        ⎪
  ⎪                                                                 ⎪
  ⎪            ,IGETFC=⎧ NO  ⎫,IGETIC=⎧ NO  ⎫                        ⎪
  ⎪                    ⎩ YES ⎭        ⎩ YES ⎭                        ⎪
  ⎪                                                                 ⎪
  ⎪            ,ICFD=⎧ NO  ⎫                                         ⎪
  ⎩                  ⎩ YES ⎭                                         ⎭
,RC=OLD / NEW
[,VTSUCBA=addr]
[,TIMER=value]
,PARMOD=24 / 31
[,MF=C / (C,pre) / (E,...) / (D,pre) / D / L]
```

record
Symbolic address of the field to which the data record to be read is transferred. The field starts with a record length field. Record format:

Byte 0-1:	record length + 4-byte record length field
Byte 2-3:	reserved
Byte 4-n:	data record.

Example

```
RECORD    DS      0CL74
LENGTH    DS      CL2
RESERV    DS      CL2
DATA      DS      CL70
```

error
Symbolic address (name) in the user program to which a branch is made:

- when an error occurs (read error, end of file,...)
- if operand A is set.

If an error occurs, register R14 contains the address of the next instruction after the **RDATA** macro call. The error code is transferred in register R15.
31-bit interface: if error = 0 (address X'00..0') is specified, the program continues with the next instruction after the **RDATA** macro call.

length
Specifies the size of the read input area (“record”) including 4 bytes for the record length field. The maximum size permitted is 32767 bytes. If this operand is omitted, the length attribute of “record” is assumed.

edit
Specifies the edit option for a message input from the terminal. This operand is not necessary when standard functions (all edit bits = 0) are used, when a MODE specification is made or when the VTSU control block is used. Direct specification (X'xx') enables only the first edit byte for input to be set to the same meaning as specified in the **CUPAB** macro description.

Notes

> This parameter only continues to be supported for reasons of compatibility.
> The edit options should be controlled via MODE specifications (see the MODE operand) or via the VTSU control block (see the VTSUCBA operand).

A
The user program is notified of the initial standard assignment and each subsequent assignment to SYSDTA. This notification occurs via the error address (“erraddr”) as soon as the read operation has been completed. The assignment code is stored as follows:

24-Bit interface:	in the leftmost byte of register R15.
31-Bit interface:	in the CURAIND field of the operand list.

KEYLEN=
Determines whether or not the ISAM key length is to be stored. This operand is evaluated only if SYSDTA is assigned to an ISAM file.

> **<u>N</u>**
> The length of the ISAM key is not stored.
>
> **Y**
> The length of the ISAM key decremented by 1 is stored as follows:
> - 24-bit interface: in the rightmost byte of register R0.
> - 31-bit interface: in the CURKEYL field of the operand list.

KEYOUT=
Determines whether the record is to be transferred with or without the ISAM key. This operand is evaluated only if SYSDTA is assigned to an ISAM file.

N
The ISAM key is not to be removed.

Y
The ISAM key is removed.

KEYPOS=
Determines whether or not the ISAM key position is to be stored. This operand is evaluated only if SYSDTA is assigned to an ISAM file.

N
The ISAM key position is not stored.

Y
The ISAM key position decremented by 1 is stored as follows:
24-bit interface:

- in the middle two bytes of register R0 if KEYLEN=Y is specified
- in the two rightmost bytes of register R0 if KEYLEN=N is specified.

31-bit interface: in the CURKEYP field of the operand list.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A prefix (pre = 1..3 letters) can be specified in the C form and D form of the macro, as shown in the macro format.
Default value: pre = CUR

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Note

Any character string beginning with “Y” has the same meaning as the “Y” entry. Any character other than “Y” is interpreted as “N”; similarly any character string beginning with a character other than “Y” has the same effect as an “N” entry and gives rise to an MNOTE message.

The following operands are interpreted only if SYSDTA is assigned to the terminal. The MODE specifications together with the edit options continue to be supported for compatibility reasons only. They are now summarized in the VTSU control block (VTSUCB, see the **VTSUCB** macro).

MODE=COMP
Specifies compatible mode. Symbolic operands enable the user program to use all the edit options as defined in the edit option table (see the **CUPAB** macro). Any specifications made in the “edit” operand are ignored. Control characters in the text are passed unchecked to the user program. This operating mode is compatible with previous versions (compatible terminals supported: 8103, 8110, 8150, 8152, 8161).
This mode is interpreted as MODE=LINE for the devices 3270, 8160, 8162, 9749, 975x. All edit options other than ILCASE, IGETBS are rejected (RC: X'08').

MODE=LINE
If SYSDTA is a terminal, it is treated as a logical line terminal. The message may be structured with logical control characters (see the **VTCSET** macro). A device-specific message header is not provided. The operands for message editing are interpreted.
If SYSDTA is a cataloged file, i.e. not a terminal, the operands for message editing are not interpreted. For example, lowercase letters are not converted into uppercase letters if SYSDTA is a cataloged file and ILCASE has the NO value (default value). Logical control characters in the message are also not interpreted.

ICFD=
Specifies whether confidential data is to be protected.

<u>NO</u>
No precautions are to be taken to protect confidential data.

YES
The input data is confidential and is to be invisible at the terminal. Depending on the terminal used, this is implemented by blanking or clearing the screen or by overwriting the input line in the case of printer terminals.

IGETBS=
Determines whether underline characters (X'6D') are passed to the user program. This operand should be specified only for 8103 Data Display Terminals.

<u>NO</u>
Underline characters are not passed to the user program. Instead, the system performs the correction function.

YES
The underline characters (X'6D') are passed to the user program without being evaluated by the system.

IGETFC
Determines whether a function key code is to be transferred.

NO
No function key code is to be transferred.

YES
The 5th byte of the input area is to contain the standardized function key code. This code identifies the terminal key used to initiate data transfer (for standardized function key codes see the appendix).

IGETIC=
Determines whether the input source is to be changed.

NO
The input source is not to be changed.

YES
Input is to be from the connected ID card reader. The input data can consist only of ID card information or of the short code K14. This entry is permitted only for 8160, 9749, 975x and 3270 Terminals with a defined ID card reader (see also the **TSTAT** macro, TYPE=TCHAR). The IGETIC operand is ignored if ICFD=YES is also specified or if no ID card reader is connected.

IHDR=
Specifies how the message header is to be handled.

NO
The message header is not transferred to the user program.

YES
The entire message header is transferred to the user program. In the case of 3270 Terminals, the message header consists of the application ID (AID byte) and the 2-byte cursor position.

ILCASE=
Determines whether a distinction is to be made between uppercase and lowercase letters.

NO
All lowercase letters are transferred to the user program in uppercase format.

YES
Lowercase letters are also transferred to the user program.

ILINEND=
Specifies how carriage return and line feed characters are to be handled.

NO
The carriage return and line feed characters are not passed to the user program.

YES
The carriage return and line feed characters are passed to the user program.

ITRSUP=
Specifies whether or not the translation from device code to EBCDIC is to be suppressed.

NO
Translation from device code to EBCDIC is not suppressed. The user program receives the message in EBCDIC.

Exception
In the case of the 8161 Data Display Terminal, the message header is always supplied in the device code.

YES
Translation from device code to EBCDIC is suppressed. The user program receives the message in the device code. This entry is not valid for the 8161 Data Display Terminal.

RC=
Determines where the return code is to be stored.
This operand is permitted only for a 31-bit interface.

OLD
The return code is stored in the rightmost byte of register R15.

NEW
The return code is stored both in register R15 and in the standard header. All 4 bytes of register R15 are allocated for evaluation. A 4-byte return code is supplied only if SYSDTA reads from the data display terminal. In all other cases, only a 1-byte return code is supplied, irrespective of the return code value.

TIMER=value
Defines a maximum wait time for input. If no input is received within the defined wait time, a return code is issued. This operand may be specified only for the 31-bit interface.

value
Wait time of 10 to 3600 seconds. The default value is UNLIMITED, i.e. no timer is used.

VTSUCBA=addr
Defines the address of a VTSUCB generated with MF=L. This When the VTSUCBA operand is used, the MODE operand and the following EDIT options are ignored (the operand value is set to X'FF' in the parameter list). This means that all desired EDIT options must be specified in the VTSUCB.
This operand may be specified only for the 31-bit interface.
The VTSUCB is not used by default.

addr
Symbolic address (name) of the VTSUCB.

Macro format 2 and description of operands

RDATA
(1) [,PARMOD=24 / 31]

(1)
Register R1 contains the operand list address. The list must be aligned on a word boundary.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Layout of the data area

Interface	Byte	Contents
31-bit interface	0 - 7	Standard header. For structure see section “Standard header” on page 43 The initialization values should be taken from an operand list generated with MF=L. No return code is transferred in the standard header if RC=OLD.
	8 - 11	Address to be branched to if an error occurs (operand “error”).
	12 - 15	Address of the field to which the data record read is transferred (operand “record”).
	16	Input edit byte 1
	17	Input edit byte 2
	18	Assignment code for SYSDTA
	19	Flag indicating use of VTSUCB and return code handling
	20 - 21	Maximum length of the data record to be read (operand “length”)
	22	Flag for edit byte of ISAM key
	23	SYSDTA assignment indicator (Bit 2^0 =1 ≙ operand A)
	24 - 25	Position of the ISAM key
	26 - 27	Length of the ISAM key
	28 - 31	Address of the VTSUCB
	32 - 33	Timer values
	34 - 35	Reserved (X'00000000')
24-bit interface	0	Input edit byte 1
	1 - 3	Address of the field to which the data record read is transferred (operand ' record')
	4	Flag
	5	Input edit byte 2
	6 - 7	Maximum length of the data record to be read (operand “length”).
	8	SYSDTA assignment indicator (Bit 2^0 =1 ≙ operand A)
	9 - 11	Address to be branched to if an error occurs (operand “error”).

- When using the 24-bit interface, the values for input edit byte 1/2 should be taken from the table specified with the **CUPAB** macro.
- When using the 31-bit interface, they should be taken from a data list generated with MF=C/D.
- Assignment code for SYSDTA

  24-bit interface: The assignment code is transferred in the leftmost byte of register R15

  31-bit interface: The assignment code is transferred in the CURAIND field of the **RDATA** operand list.

Code values and their meaning

Value	Meaning
X'00'	Assignment for SYSDTA is unchanged
X'04'	SYSDTA is assigned to a SAM file
X'08'	SYSDTA is assigned to an ISAM file
X'14'	SYSDTA is assigned to a terminal
X'18'	SYSDTA is assigned to an S variable
X'20'	SYSDTA is assigned to a PLAM library element

– Flag byte and its meaning:
  Bit 2^7 = 1/0 corresponds to KEYOUT=Y/N.
  Bit 2^6 = 1/0 corresponds to KEYPOS=Y/N.
  Bit 2^5 = 1/0 corresponds to KEYLEN=Y/N.

Return information and error flags

- Where possible, the system corrects any edit options that are invalid for a particular device or for the operating mode MODE=.
- During macro processing, register R1 contains the operand list address.

if PARMOD=24:

R15:

b	b	0	0	0	0	a	a

A return code relating to the execution of the RDATA macro is transferred in register R15, where: aa = Maincode; bb = assignment code if operand A is specified in conjunction with the 24-bit interface; in all other cases bb = X'00'.

X'aa'	**Meaning**
X'00'	Normal termination
X'04'	Unrecoverable error
X'08'	Operand error
X'0C'	Record truncated. Record length > specified length
X'10'	End of file (EOF)
X'14'	SYSDTA not assigned
X'18'	Error during volume access
X'20'	Invalid edit option byte: error corrected by the system
X'38'	Problem in connection with POSIX

if PARMOD=31, RC=OLD:

Return codes that may occur in addition to those described under PARMOD=24 are X'24' (Error in VTSUCB) and the return codes which, in accordance with conventions, apply to all macros (see the table “Standard return codes” on page 43).

if PARMOD=31, RC=NEW:

The return codes are entered both in the standard header and in register R15.

Standard header:

c	c	b	b	a	a	a	a

A return code relating to execution of the RDATA macro is transferred in the standard header: aaaa=Maincode; bb=SUBCODe1, cc=Subcode2

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function processed successfully
X'00'	X'00'	X'0014'	Function processed successfully, but SYSDTA not assigned
X'00'	X'00'	X'0018'	Function processed successfully, but error occurred when accessing volume
X'00'	X'00'	X'0020'	Function processed successfully. Operand error corrected via TIAM/VTSU
X'00'	X'01'	X'0008'	Operand error not corrected
X'07'	X'01'	X'0008'	Operand error not corrected: the RESERVED fields are not 0
X'08'	X'01'	X'0008'	Operand error not corrected: value of TIMER operand not in permitted range (10 - 3600 seconds)
X'00'	X'20'	X'0004'	Internal error
X'02'	X'20'	X'0004'	Internal error: BCAM message lost
X'05'	X'20'	X'0004'	Internal error: input message too long
X'06'	X'20'	X'0004'	Internal error: negative transport acknowledgment
X'00'	X'40'	X'0004'	Input/output aborted
X'00'	X'40'	X'000C'	Input record length > specified length: input record truncated
X'00'	X'40'	X'0010'	End of file (EOF)
X'00'	X'40'	X'0034'	Timeout (no input received within specified wait time)
X'01'	X'80'	X'0004'	Internal BCAM bottleneck
X'09'	X'80'	X'0038'	Error in connection with POSIX: Input/output serialization error
X'0A'	X'40'	X'0038'	Error in connection with POSIX: If the LOGON task is in system mode, no inputs/outputs of processes generated with fork() are possible
		X'24'	VTSU error. In addition to main code (rightmost byte), see error information in VTSUCB header

Notes on the macro call

- Truncation occurs if the record to be transferred is longer than specified in the length operand. The record is only transferred to the read input area in accordance with the length specified in the length operand; the remainder of the record is lost. If the record is shorter than the read input area, it is entered left-justified and the remainder is not filled with blanks. The program continues without an error flag.
- The end-of-file condition (EOF) can occur in the following ways:
  - Interactive mode:
    Activate the ESCAPE function (K2 key). Job processing switches over to command mode. Then issue the EOF command followed by RESUME-PROGRAM.
  - Procedure or batch mode:
    The (system) files SYSDTA and SYSCMD are assigned to each other and a data record starting with a slash is read.
    Exceptions:
    - the HOLD-PROGRAM command has been issued.
    - the slash has been replaced by symbolic operands.
    - the record starts with two consecutive slashes (e.g. SDF statement).
  - The (system) files SYSDTA and SYSCMD are not assigned to each other:
    The EOF command is detected in columns 1-4 of the record.

For **examples**, see section “S-type macros” on page 29.

RDUID – Read user ID

General

Application area:	Requesting and accessing lists and tables; see page 155
Macro type:	Type S, MF format **2**: standard/C/D/L/E form; see page 29

Macro description

The **RDUID** macro transfers to a user program in its operand list the user ID and the account number of the job under which it is running (see the layout of the operand list after the operand description).

Macro format and description of operands

RDUID
MF=S / E / L / C / D [,PARAM=addr / (r)] ,PREFIX=S / p ,MACID=RMR / macid

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form or D form of the macro and additionally a MACID in the C form (see section “S-type macros” on page 29).

Return information and error flags

Standard header:

0	0	b	b	a	a	a	a

The following return code relating to the execution of the RDUID macro is transferred in the standard header (bb=Subcode1, aaaa=Maincode):

X'bb'	X'aaaa'	Meaning
X'00'	X'0000'	Function successfully executed
X'20'	X'00FF'	System erro

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

The calling program is terminated when the following errors occur:

- The data area is not assigned to the caller.
- The data area is not aligned on a word boundary.
- The data area is protected against write access.

Layout of the data area for RDUID MF=C

```
1          FHDR  MF=(C,SRMR),EQUATES=NO
2          DS    0A
2 SRMRFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 SRMRIFID DS    0A              0   INTERFACE IDENTIFIER
2 SRMRFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 SRMRFCT  DS    AL1             2   FUNCTION NUMBER
2 SRMRFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 SRMRRET  DS    0A              4   GENERAL RETURN CODE
2 SRMRSRET DS    0AL2            4   SUB RETURN CODE
2 SRMRSR2  DS    AL1             4   SUB RETURN CODE 2
2 SRMRSR1  DS    AL1             5   SUB RETURN CODE 1
2 SRMRMRET DS    0AL2            6   MAIN RETURN CODE
2 SRMRMR2  DS    AL1             6   MAIN RETURN CODE 2
2 SRMRMR1  DS    AL1             7   MAIN RETURN CODE 1
2 SRMRFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1 SRMRUID  DC    CL8' '              USERID
1 SRMRACC  DC    CL8' '              ACCOUNT NUMBER
1 SRMR#    EQU   *-SRMRFHE           LENGTH OF RDUID PARAMETER BLOCK
```

RELBF – Release receive queue

General

Application areas::	Intertask communication; see page 76 Communication; see page 163
Macro type:	Type O; see page 28

Macro description

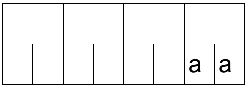
Users participating in intertask communication (ITC) can delete the first message in their receive queue by means of the **RELBF** macro.
Users who want to analyze their receive queue may request the first message from a specified sender or the first message in the queue (FIFO). If they proceed according to the FIFO principle, they must delete the first message in the queue before accessing the next one. If the first message was not deleted implicitly when it was requested (REVNT...REL=YES), it may be deleted explicitly by means of the **RELBF** macro.
A user who wants to delete the entire receive queue but does not yet want to terminate ITC participation (CLCOM) may keep issuing the **RELBF** macro in a loop until the return code indicates that the queue is empty.

Macro format and description of operands

RELBF

Return information and error flags

R15:



A return code relating to the execution of the RELBF macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X' 00'	The first message in the receive queue has been deleted
X' 04'	The receive queue was empty
X' 08'	The task of the calling program is not an ITC participant

RELM – Release memory

General

Application area:	Working with virtual memory; see page 55
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

Macro description

The **RELM** macro releases a contiguous area of the caller's class 6 memory. Memory is released in multiples of one page (4 Kb) . It is permissible to release memory space which has not previously been requested with **REQM**. Memory areas previously requested by various **REQM** macro calls can be released by means of a single **RELM** call if they are contiguous.

Macro format and description of operands

```
RELM
----------------------------------------
 [{ number }],{ page }
  { (r)    }   { (r)  }

[,PARMOD=24 / 31]
,MF=S / (E,...) / L
```

number
Number of pages (4 Kb) to be released.
Default value: number = 1.

(r)
Register containing “number”.

page
Page number of the first page (4 Kb) of the area to be released.

(r)
Register containing “page”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:

0	0	0	0	0	0	a	a

A return code relating to the execution of the RELM macro is transferred in register R15.

X'aa'	Meaning
X'00'	Memory space was released
X'04'	Function only partially executed. The area to be released also contains nonrequested pages. The address of the first nonrequested page is transferred in register R1. All pages in the area up to this address were released, all pages after it are in the same state as before the RELM call
X'0C'	Function not executed. – Invalid operand list address – Error in operand list structure – The area to be released lies (partly) outside the class 6 memory – The area to be released (partly) overlaps a memory pool – The area to be released (partly) overlaps a FASTPAM ENVIRONMENT/ IOAREA POOL. It is not possible to release the area until the corresponding FASTPAM disable function has been executed – The area to be released (partly) overlaps a DIV window It is not possible to release the area until the correspondinng DIV unmap function has been executed – The access key of the caller at the time of the RELM call does not match the area to be released

If the 31-bit interface is used:

- In the event of errors in the initialization of the standard header, the return codes X'0001FFFF' / X'0003FFFF' / X'0004FFFF' are additionally transferred; see the table “Standard return codes” on page 43.
- No return codes are transferred in the standard header.

Examples

1. Let the user program occupy pages 1 - 12.
   RELM 3,10 releases three pages starting at page 10, i.e. pages 10, 11 and 12.
2. Let the user program occupy pages 1 - 6.
   RELM ,5 releases one page starting at page 5, i.e. page 5 is released.

RELMP – Release pages in memory pool

General

Application area: Memory pools; see page 55
Macro type: Type S, MF format **1**: standard/L/E form; see page 29

A user can explicitly request (**REQMP**) and also release (**RELMP**) contiguous memory space in a memory pool. The following applies to the release of memory space in a memory pool:

- the caller must be a pool participant (**ENAMP**),
- it is irrelevant which of the pool participants requested the memory space and in what units it was requested.

Macro description

The user can release contiguous memory space in a memory pool with the **RELMP** macro. The release is carried out in memory pages (4K).

Notes

- A memory pool is accessed via the pool name or via its ID (see **ENAMP**).
- The release of pool pages does not change the size of a memory pool as specified in the **ENAMP** macro.
- The memory space to be released need not have been allocated as a contiguous memory area.
- **RELMP** is rejected if the memory pool is write-protected (**CSTMP** macro).

Macro format and description of operands

```
RELMP
--------------------------------------------------------------------------------------------
⎧ ⎧ MPNAME=name                                        ⎫             ⎧ LOCAL      ⎫   ⎫
⎪ ⎨                                                    ⎬[,SCOPE=     ⎨ GROUP      ⎬]  ⎪
⎨ ⎪ MPNAMAD=⎧ addr ⎫[,MPNAMLN=⎧ length ⎫]              ⎪             ⎪ USER_GROUP ⎪   ⎬
⎪ ⎩         ⎩ (r)  ⎭          ⎩ (r)    ⎭               ⎭             ⎩ GLOBAL     ⎭   ⎪
⎩ MPID=addr / (r)                                                                      ⎭
[,ADDR=addr / (r)]
[,BSIZE=number / (r) / ALL]
[,PARMOD=24 / 31]
[,MF=L / (E,..)]
```

MPNAME=
Defines the name of the memory pool (note the connection with the SCOPE operand).

name
Name of the memory pool.

MPNAMAD=
Specifies the address of the field containing “name” (note the connection with the SCOPE operand).

addr
Symbolic address (name) of the field.

(r)
Register containing the address value “addr”.

MPNAMLN=
Specifies the length of the memory pool name specified under MPNAMAD. If omitted: length attribute of the “addr” field, or 54 bytes if MPNAMAD=(r) was specified.

length
Length in bytes

(r)
Register containing “length”.

SCOPE=
Defines the scope (authorized users) of the memory pool. This specification is used to identify the memory pool uniquely and must always be entered in conjunction with the MPNAME or MPNAMAD operand.

LOCAL
The memory pool is only used by the user who created it.

GROUP
Memory pool users can be all tasks with the ID of the user that created the memory pool.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.
The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
Users can be all the tasks running in the system.

MPID=
Specifies the address of a field (length = 4 bytes) with the ID for the memory pool (see **ENAMP**). The ID provides unique identification of the memory pool; this operand speeds up processing.

addr
Symbolic address (name) of the field containing the ID.

(r)
Register containing the address value “addr”.

ADDR=
Specifies the start address of the memory area to be released. The address must be aligned on a 4K boundary. The entire area must be in the specified memory pool.

Note

The ADDR operand is mandatory unless BSIZE=ALL is specified.

addr
Start address.

(r)
Register containing the start address.

BSIZE=
Specifies the size in (4K) memory pages of the memory area to be released.
Default value: BSIZE=1.

Note
The BSIZE=0 operand is permitted; memory space is not released.

number
Number of memory pages.

(r)
Register containing “number”.

ALL
All the memory pages requested for the memory pool are released.
This operand can also be executed in register “r” in the form C' ALL'.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists use 24-bit addresses.
(Address space ≤ 16 Mb).

31
The 31-bit interface is generated.
Data lists use 31-bit addresses (address space ≤ 2 Gb) and start with the standard header.

Return information and error flags

After macro processing, register R1 contains the operand list address.

R15:

b	b	0	0	0	0	a	a

A structured return code (aa = primary return code, bb = secondary return code) relating to the execution of the RELMP macro is transferred in register R15.
aa = X'00': normal execution,
aa = X'04': function was not carried out.

X'bb'	X'aa'	Meaning
X'00'	X'00'	Normal execution. Memory pages were released
X'18'	X'00'	Normal execution. Memory pages were released; not all the pages in the specified area had been requested
X'04'	X'04'	Function was not executed. The caller is not a memory pool participant (no ENAMP macro)
X'18'	X'04'	Function was not executed; invalid memory area: – The start address or an address of the specified memory area is not within the memory pool – The start address is not aligned on a 4K boundary – The specified memory area completely or partially overlaps an area used by DIV or FASTPAM
X'1C'	X'04'	The function was not executed; operand error: – Invalid operand list address – Error in operand list structure – Invalid address for MPNAMAD or MPID in the operand list – Memory pool designation: – name contains invalid characters – invalid length specification (MPNAMLN) – MPNAMLN was specified but MPNAMAD was not – MPNAME, MPNAMAD and MPID were not specified – SCOPE specified but MPNAME/MPNAMAD not specified – designation is not unequivocal; more than one of operands MPNAME/ MPNAMAD/MPID was specified – Invalid SCOPE specification – Invalid BSIZE specification – Neither the ADDR nor BSIZE=ALL operands were specified – An invalid register was specified (R1) – PARMOD=24 was specified in conjunction with 31-bit addressing mode (AMODE31) – SCOPE=USER_GROUP was specified, although SRPM is not available in the system – The release of pages not in a memory pool is no longer supported

X'bb'	X'aa'	Meaning
X'24'	X'04'	Function was not executed; authorization error: – The memory pool is write-protected. – The memory pool was protected by a privileged user against release. Repeat the macro, if required – The caller is not authorized to release memory pages from a privileged or a class 5 memory pool – The access key at the time of the RELMP call does not match the access key valid when the memory pool was created

31-bit interface:

- In the event of errors in the initialization of the standard header, the return codes X'0001FFFF' / X'0003FFFF' / X'0004FFFF' are additionally transferred in register R15; see table “Standard return codes” on page 43.
- No return codes are transferred in the standard header.

REQM – Request memory

General

Application area:	Working with virtual memory; see page 55
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

Macro description

REQM requests that a contiguous memory area be assigned to the user program. This area can be released again with **RELM**. Memory is requested in multiples of one page (4 Kb). The memory allocated is always virtual. Each byte of the allocated page(s) is overwritten with X'00' (except if the page had already been requested).

Macro format and description of operands

```
REQM
-------------------------------------------------------------
                  ⎧ page                   ⎫
                  ⎪ (r)                    ⎪
                  ⎪                        ⎪
  ⎧ number ⎫      ⎪        ⎧ RES    ⎫      ⎪
 [⎨        ⎬][,   ⎨        ⎪ ANY    ⎪      ⎬ ]
  ⎩ (r)    ⎭      ⎪ LOC=   ⎨ BELOW  ⎬      ⎪
                  ⎪        ⎩ ABOVE  ⎭      ⎪
                  ⎩                        ⎭

,ALIGN=4KB / HW_PAGE
[,PARMOD=24 / 31]
,MF=S / (E,..) / L
```

number
Number of pages (4 Kb) to be requested.
Default value: number = 1.

(r)
Register containing “number”.

page
Page number (4 Kb) at which allocation is to begin.
Default setting: Smallest page number of the first area in the caller's class 6 memory which is sufficiently large and has not yet been allocated. The area may be aligned during function execution (e.g. on a multiple of 16 pages) if the number of pages requested exceeds a specific threshold.

Note

> This operand should be specified only after an inquiry as to the size and location of the caller's class 6 memory has been made by means of the **MINF** macro.

(r)
Register containing “page”.

LOC=
Identifies that part of class 6 memory where the memory pages are to be reserved (below or above the 16-Mb boundary).
The operand is ignored if the 24-bit interface is used.

> **RES**
> Default setting: memory pages are reserved in that part of class 6 memory where the macro call is issued.
>
> **ANY**
> Memory pages are reserved as follows, depending on the addressing mode in use:
> - below the 16-Mb boundary if 24-bit addressing mode is activated,
> - above or below the 16-Mb boundary if 31-bit addressing mode is activated.
>
> **BELOW**
> Memory pages are reserved below the 16-Mb boundary.
>
> **ABOVE**
> Memory pages are reserved above the 16-Mb boundary.

ALIGN=
Specifies the alignment of the requested area.

> **4KB**
> The requested area is aligned on 4-Kb boundary.
>
> **HW_PAGE**
> The requested memory area begins on a page boundary determined by the hardware. In this case, the operands “number” and “page” must be used in such a way that a multiple (in 4 Kbyte units) of hardware page sizes is achieved.
>
> *Note*
>
> > The hardware page size can be obtained by the **NSIINF INFO=PAGESIZE** macro.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Return information and error flags

During macro processing, register R1 contains the operand list address.
After successful execution of the macro, the start address of the allocated area is stored in register R1.

R15:

0	0	0	0	0	0	a	a

A return code relating to the execution of the REQM macro is transferred in register R15.

X'aa'	Meaning
X'00'	The request has been processed successfully. Register R1 contains the address of the first page even if the page had already been requested
X'04'	Function not executed. – Insufficient contiguous free memory space is available in the address space – Insufficient free space is available in the paging area
X'0C'	Function not executed – Invalid operand list address – Error in operand list structure – The requested number of pages exceeds – the value for the ADDRSPACE parameter in the user catalog (see output of SHOW-USER-ATTRIBUTES command) – the area available below or above 16 Mb (see output of MINF macro) – The requested area lies (partly) outside the class 6 memory – The requested area (partly) overlaps a memory pool – The access key of the caller at the time of the REQM call does not match the requested area – The specified number or page parameter is not a multiple of the hardware page size in 4 Kb units and ALIGN=HW_PAGE is specified.

For the 31-bit interface:

- In the event of errors in the initialization of the standard header, the return codes X'0001FFFF' / X'0003FFFF' / X'0004FFFF' are additionally transferred; see table “Standard return codes” on page 43.
- No return codes are transferred in the standard header.

REQMP – Request pages in memory pool

General

Application area:	Memory pools; see page 55
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

The user can explicitly request (contiguous) space in a memory pool, where:

- the caller must be a pool participant (**ENAMP**).
- any pool participant can release the allocated memory area.
- any pool participant can access the allocated memory area.

Macro description

A contiguous memory area in a memory pool is requested with the **REQMP** macro. The allocation is carried out in memory pages (4Kb). The bytes of the requested page(s) are overwritten with X'00' (unless the page(s) have been requested before).

Notes

- A memory pool is addressed via a pool name or its ID (see **ENAMP**).
- **REQMP** is rejected if the memory pool is write-protected (**CSTMP** macro).
- Resident memory pool: **REQMP** is rejected if the number of pages requested exceeds the value of the RESIDENT-PAGES operand in the START-PROGRAM or LOAD-PROGRAM command.
- An inquiry as to the size of the memory pool and the status of memory pages (allocated or not) can be made by means of **MINF**.

Macro format and description of operands

```
REQMP
-------------------------------------------------------------------------------------------
{  {  MPNAME=name                                              }             { LOCAL      }   }
{  {                                                           }             { GROUP      }   }
{  {  MPNAMAD= { addr } [,MPNAMLN= { length } ]                } [,SCOPE=    { USER_GROUP } ] }
{  {           { (r)  }            { (r)    }                  }             { GLOBAL     }   }
{                                                                                             }
{  MPID=addr / (r)                                                                            }

[,ADDR=addr / (r)]
[,BSIZE=number / (r)]
,ALIGN=4KB / HW_PAGE
[,PARMOD=24 / 31]
[,MF=L / (E,..)]
```

MPNAME=
Defines the name of the memory pool (Note the connection with the SCOPE operand).

name
Name of the memory pool.

MPNAMAD=
Specifies the address of the field containing “name” (note the connection with the SCOPE operand).

addr
Symbolic address (name) of the field.

(r)
Register containing the address value “addr”.

MPNAMLN=
Defines the length of the name specified under MPNAMAD. If omitted: length attribute of the “addr” field, or 54 bytes if MPNAMAD=(r) was specified.

length
Length in bytes.

(r)
Register containing “length”.

SCOPE=
Defines the scope (authorized users) of the memory pool. This specification is used to identify the memory pool uniquely and must always be entered in conjunction with the MPNAME or MPNAMAD operand.

LOCAL
The memory pool is only used by the user who created it.

GROUP
Memory pool users can be all tasks with the ID of the user that created the memory pool.

USER_GROUP
All the tasks whose user IDs belong to the same user group as the user ID of the creating participant can be participants.
The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
Users can be all the tasks running in the system.

MPID=
Specifies the address of a field (length = 4 bytes) with the ID for the memory pool (see **ENAMP**). The ID provides unique identification of the memory pool; this operand speeds up processing.

addr
Symbolic address (name) of the field containing the ID.

(r)
Register containing the address value of the field.

ADDR=
Specifies the start address of the memory area to be allocated. The address must be aligned on a 4Kb boundary. The entire area must be in the specified memory pool.
Default value: the first unused, contiguous area in the memory pool or in user memory below the 16-Mb boundary. The area may be aligned on a specific boundary (64Kb or 1-Mb boundary) if the number of pages requested exceeds a defined threshold.

addr
Start address.

(r)
Register containing the start address.

BSIZE=
Specifies the size of the requested memory area in memory pages (4Kb).
Default value: BSIZE=1; this value is also assumed if BSIZE=0 was specified.

number
Number of memory pages.

(r)
Register containing “number”.

ALIGN=
Specifies the alignment of the requested area.

4KB
The requested area is aligned on 4-Kb boundary.

HW_PAGE
The requested memory area begins on a page boundary determined by the hardware. In this case, the operands BSIZE (in 4 Kbyte units) and ADDR (as the start address) must be used in such a way that a multiple of hardware page sizes is achieved.

Note

The hardware page size can be obtained by the **NSIINF INFO=PAGESIZE** macro.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated. If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists use 24-bit addresses
(address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists use 31-bit addresses
(address space ≤ 2 Gb) and start with the standard header.

Return information and error flags

After successful execution of the function, register R1 contains the start address of the allocated memory area.

R15:

b	b	0	0	0	0	a	a

A structured return code (aa = primary return code, bb = secondary return code) relating to the execution of the REQMP macro is transferred in register R15.
aa = X'00': normal execution,
aa = X'04': function was not carried out.

X'bb'	X'aa'	Meaning
X'00'	X'00'	Normal execution. Memory was allocated
X'18'	X'00'	Normal execution. At least one memory page was already allocated within the requested area. The remaining memory pages were allocated
X'04'	X'04'	Function was not executed. The caller is not a memory pool user (no ENAMP macro)
X'14'	X'04'	Function was not executed. Insufficient memory space: – Memory pool does not have this amount of free contiguous memory space available – Resident memory space: the request exceeds the value of the RESIDENT-PAGES operand in the START-PROGRAM or LOAD-PROGRAM command – BSIZE or ADDR is not a multiple of the hardware page size and ALIGN=HW_PAGE is specified.
X'18'	X'04'	Function was not executed. Invalid memory area: – Start address or an address of the specified memory is not within the memory pool – The start address is not aligned on a 4K boundary
X'24'	X'04'	Function was not executed. Authorization error: – The memory pool is write-protected – The caller is not authorized to request memory pages from a privileged or a class 5 memory pool – The access key at the time of the RELMP call does not match the access key valid when the memory pool was created

X'bb'	X'aa'	Meaning
X'1C'	X'04'	The function was not executed. Operand error: – Invalid operand list address – Error in operand list structure – Invalid address for MPNAMAD or MPID in the operand list – Memory pool designation: – name contains invalid characters – invalid length specification (MPNAMLN) – MPNAMLN was specified but MPNAMAD was not – MPNAME, MPNAMAD and MPID were not specified – SCOPE was specified but MPNAME/MPNAMAD was not – designation not unequivocal: more than one of the operands MPNAME/MPNAMAD/MPID was specified – Invalid SCOPE operand – Invalid BSIZE operand – Invalid register (R1) specified – SCOPE=USER_GROUP was specified although SRPM is not available in the system – PARMOD=24 was specified in conjunction with 31-bit addressing mode (AMODE31) – Requests for pages not in a memory pool are no longer supported

31-bit interface:

- In the event of errors in the alignment or initialization of the standard header, the return codes X'0001FFFF' / X'0003FFFF' / X'0004FFFF' are additionally transferred in register R15; see table “Standard return codes” on page 43.
- No return codes are transferred in the standard header.

RETCO – Return from contingency process

General

Application area:	Contingency processing; see page 110
Macro type:	Type S, MF format **1**: standard/E/L form; see page 29

Macro description

The **RETCO** macro enables return from a contingency process. Control is returned to a basic task or contingency process having the same or a lower priority level.
RETCO must not be issued in the basic task (abnormal program termination with error message `ETMEV03`).

Macro format and description of operands

RETCO
[MF=L / E]

MF
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

For **examples**, see section “Contingency processes” on page 110 and the **POSSIG** macro description (page 736).

RETRN – Load return with register

General

Application area: Starting, interrupting and terminating; see page 72
Macro type: Type O; see page 28

Macro description

The **RETRN** macro serves to reload saved register contents, to return from a subroutine to the main program and to pass on a return code, if desired.
The **RETRN** macro is issued at the end of a subroutine if, at the beginning of that subroutine, the registers of the main program were saved by means of the **SAVE** macro. Except for register 13, all general register contents can be buffered and reloaded. The address of the save area supplied by the main program must be contained in register 13 (see the **SAVE** macro).

Macro format and description of operands

RETRN
[(r1[,r2])] [,T] [,RC=rc / (15)

r1
Specifies a register (general register) that is to be loaded from the save area. Numbers between 0 and 15 (except 13) may be specified for “r1”. The parentheses are optional.

r1,r2
Specifies a consecutive series of general registers which are to be reloaded. The range r1, r2 must not contain register 13, but it may cross the register 15/register 0 boundary.
Example: All registers except register 13, i.e. registers 0 to 12, 14 and 15, are loaded from the save area by means of the entry (14,12).

T
Is designed for compatibility with TOS language processors and is used to interrupt forward chaining in nested programs. After the registers have been reloaded, a “1” is set in the lowest-order bit in word 3 of the buffer.

RC=
Specifies a return code that is passed on to the main program in the rightmost three bytes of register 15.

rc
The value “rc” must be a decimal number or an absolute expression.

(15)
Specifies that register 15 contains a return code in the rightmost three bytes. In this case register 15 is not reloaded with the original contents of the buffer.

Functional description

The **RETRN** macro loads the specified registers from the save area and branches to the main program. For the call of the **RETRN** macro it is required:

- that the subroutine saved the register contents at the beginning by means of the **SAVE** macro;
- that the address of the save area has been loaded to register 13;
- that the main program has loaded the return address to register 14.

If specified, the **RETRN** macro transfers a return code in register 15. This code can be freely agreed upon by the main program and the subroutine. The description of the **SAVE** macro deals with this point in detail and contains an example.

Note

Register specifications can also be entered in the form “Rn”, where n = register number.

REVNT – Receive event

General

Application areas:	Intertask communication; see page 76 Communication; see page 163
Macro type:	Type O; see page 28

Macro description

A user participating in intertask communication (ITC) can request a message and wait for its arrival by means of the **REVNT** macro.

Functional description

Each ITC participant may request a message by means of the **REVNT** macro. It can specify whether it will accept a message from any participant, or only messages from a specified sender. If the participant is waiting for a message from an unspecified sender, this message may also be from a sender that joined ITC later.

If there is not yet a message (or none from the desired sender) in the receive queue of the calling participant when the **REVNT** macro is issued, the task is interrupted by the system. The interrupted task is continued when the message is received or after the waiting time has elapsed; the length of the waiting time may be specified in the **REVNT** call. If the message has arrived in the meantime, it is transferred to the task.

The system transcribes the message from the receive queue to the destination field of the program. If there are several messages in the queue, it transcribes the first message or the first message from the desired sender. In the **REVNT** macro the user may specify whether or not the message is to be deleted after it has been transcribed from the receive queue. If the message is not deleted, it may be transcribed again by means of another **REVNT** macro.

A message may be between 8 bytes and 64 Kbytes long (including a 4-byte record length field). A user who knows the extent of messages to be exchanged when programming may define the length of the destination field accordingly. The system enters both the message and its actual length in the destination field. If the destination field cannot accommodate the entire message, the system transcribes only the first 4 bytes of the message, but it enters the complete length in the record length field.

This is flagged by means of return code X'0C'. If the message length is not known when programming, the user should issue the **REVNT** macro with the specification REL=NO. The user receives the actual length and may base the size of the destination field on that value; then the entire message may be transcribed by means of a second **REVNT** macro. This time the user may delete the message by means of REL=YES, or issue the **RELBF** macro, which clears the first message in the receive queue.

ITC linked to eventing:
The wait state, in which a task is placed after the **REVNT** call, can be avoided by linking ITC to eventing if the message is expected from an unspecified sender. In addition, waiting for an ITC message can be combined with waiting for another event (see section “Eventing” on page 94).

ITC is linked to an event item by specifying the address of the ID of the event item as an additional operand in the **REVNT** call. This address must have been defined in a previous **ENAEI** call. As a result, the task is **not** interrupted after the **REVNT** call. The caller is provided with a restricted return code specifying whether or not the **REVNT** call has been accepted. Information concerning the arrival of the message is delivered to the task by the post code as soon as the **SOLSIG** call has been processed. The **SOLSIG** call now permits the user to request a solicit signal for an event at any time after the **REVNT** call (even prior to the **REVNT** call if a contingency process has been specified). If no message has been received, the **SOLSIG** waiting time begins. The task is interrupted unless a contingency process has been specified. The **SOLSIG** waiting time is terminated by any event occurring for the specified event item. The post code indicates the class to which the event belongs.

Notes

- A task may check its receive queue, without being interrupted, by means of the **REVNT**..., WTIME=0 call.
- The limit for the waiting time (WTIME operand) is meant to prevent participants mutually blocking one another (if each is waiting for a message from the other).

Notes on linked **REVNT**:

- The event class indicated in the post code must be checked as to whether an ITC event or another event has occurred.
- No further **REVNT** macro - linked or unlinked - may be issued prior to the completion of a linked **REVNT** call (rejected with return code X'18').
- The event item must not be deleted (**DISEI** macro) prior to the completion of a linked **REVNT** call. Otherwise, the **REVNT** call cannot be completed and messages may be lost (see the note in the section “Intertask communication (ITC)” on page 85).

Macro format and description of operands
Format 1:

REVNT
destfield,length [,WTIME=seconds] ,REL=YES / NO [,NAME=sendername] [,EIID=address]

destfield

Symbolic address of the field to which the message from the receive queue is to be transferred. The field must begin on a word boundary. After the transfer its contents are as follows:

Bytes	Contents	
0 - 7	ITC name of the participant that sent the message	
8 9	length of the complete message + 4 (for SLF)	record length field (SLF)
10 11	reserved	
12 : n	message (at least 4 bytes, if the field was too small to accommodate the complete message)	

The destination field must be at least 16 bytes long and the specification in the record length field must have a value of at least 8.

length

Decimal number specifying the length of the destination field, including the 8 bytes for the sender. The value for “length” may lie between 16 and 65543. The message including the record length field may be between 8 and 65535 bytes long.

WTIME=seconds

Specifies how long a task is to wait for the message if it has not yet been received at the time when the **REVNT** macro is issued. Times from 0 to 21599 seconds are permissible. If WTIME is not specified, the task waits 600 seconds.

REL=
Specifies whether or not the message is to remain in the receive queue.

YES
The message is deleted after being transferred from the receive queue, even if there was not sufficient space available for the message in the destination field.

NO
The message remains in the receive queue.

NAME=sendername
ITC name of an ITC participant; indicates that a message should only be transferred if the specified participant is the sender.

EIID=address
This operand is required only if ITC is to be linked to eventing (see “Linking ITC to eventing” on page 81).
The operand value “address” specifies the symbolic address of a field containing the ID of the event item. The field is 4 bytes long. The system has entered the ID in this field as a result of a previous **ENAEI** macro.

Format 2:

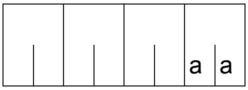
REVNT
(1)

(1)
Register 1 contains the address of an operand field with the following contents:

Bytes	**Contents**
0 - 3	address of receive field
4 - 7	length of receive field (hexadecimal)
8	X'00' : no linking to eventing X'01' : the REVNT request is linked to an event item
9 - 11	C'YES' for REL=YES X'00' C'NO' for REL=NO
12 - 15	waiting time in seconds
16 - 23	ITC name of the sender made up to 8 bytes with trailing blanks if necessary, or only blanks if no sender is to be specified
24 - 27	address of ID of event item; to be specified only if X' 01' is specified in byte 8

Return information and error flags

R15:



A return code relating to the execution of the REVNT macro is transferred in the rightmost byte of register R15.

REVNT without linkage to eventing:

X'aa'	**Meaning**
X'00'	The message has been completely transcribed
X'04'	Operand error (e.g. memory has not been allocated or is not class 6 memory). No message has been transcribed
X'08'	The calling task is not an ITC participant. No message has been transcribed
X'0C'	The destination field is too small for the complete message. Only the header and first 4 bytes have been transcribed
X'10'	Even during the waiting time no message has been received
X'18'	Processing of an earlier linked REVNT call has not been completed. The present call is rejected

REVNT with linkage to eventing:

X'aa'	**Meaning**
X'00'	REVNT call accepted
The following return codes indicate that the REVNT call does not produce an ITC event:	
X'04'	Operand error (e.g. memory is not class 6 memory or has not been allocated). The REVNT call is rejected
X'08'	The calling task is not an ITC participant. The REVNT call is rejected
X'18'	Processing of an earlier linked REVNT call has not been completed. The present call is rejected

Meaning of the post code (for linkage to eventing only):

The post code is 4 bytes long and is entered after the **SOLSIG** macro call under an address specified therein. The leftmost byte indicates the event class (see section “Eventing” on page 94). The rightmost byte contains the return code applicable to the particular event class. An ITC event has event class X'08'.

ITC post code	Meaning
X'08000000'	A linked REVNT call has been completed with the arrival of a message
X'08000004'	Operand error: The memory space for the destination field is no longer allocated (asynchronous eventing operation)
X'0800000C'	The destination field is too small for the complete message. Only the header and the first 4 bytes have been transferred
X'08000010'	The waiting time has expired and no message has been received

RPOFEI – Send POSSIG signal

General

Application area:	(Optimized) eventing; see page 94
Macro type:	Type R; see page 28

Forward eventing (FEV) is an optimized form of synchronous eventing. FEV avoids the need for repeated validation of the operands when either **POSSIG** or **SOLSIG** calls to a particular event item are repeated. Instead, an event list, EVENTLST, is set up and in order to send signals to an event item (a POSSIG function), for example, a POSSIG entry is made in the list. The entry may be explicitly deleted again (**DELFEI**).
The task of the calling program must be enabled for the event item (using **ENAEI**).

Macro description

The macro **RPOFEI** refers to a POSSIG entry in the EVENTLST and initiates the sending of a signal (event) to an event item.
This signal terminates the wait state of the requesting task, or starts a contingency routine in this task, if this was specified in a **SOLSIG** call.

Macro format and description of operands

RPOFEI
REFNUM=(r)

REFNUM=(r)
Identifies a register which (directly) holds the reference number of the POSSIG entry.

(r)
Register containing the reference number.

Return information and error flags

During the execution of the macro, register R1 contains the reference number.
Register R0 is overwritten with an internal function code.

R15:

b	b					a	a

A structured return code relating to the execution of the macro is transferred in register R15
(aa = primary return code, bb = secondary return code).

X'bb'	X'aa'	Meaning
X'00'	X'00'	Normal execution. The signal was sent
X'04'	X'04'	No action: incorrect reference number (POSSIG entry already deleted?)
X'28'	X'04'	No action: the maximum permitted number of requests in the POSSIG queue was exceeded

RSOFEI – Request POSSIG signal (event)

General

Application area:	(Optimized) eventing; see page 94
Macro type:	R-Typ; see page 28

Forward eventing (FEV) is an optimized form of synchronous eventing. FEV avoids the need for repeated validation of the operands when either **SOLSIG** or **POSSIG** calls to a particular event item are repeated. Instead, an event list, EVENTLST, is set up and in order to request signals from an event item (a SOLSIG function), for example, a SOLSIG entry is made in the list. The entry may be explicitly deleted again (**DELFEI**).
The task of the calling program must be enabled for the event item (using **ENAEI**).

Macro description

The macro **RSOFEI** refers to a SOLSIG entry in the EVENTLST and requests a signal (event) from an event item. The task of the calling program is put into a wait state if the requested signal has not yet been received, but for not longer than the duration of the specified waiting time (using the macro **DSOFEI**).

Macro format and description of operands

RSOFEI
REFNUM=(r)

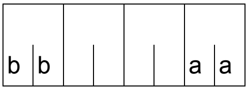
REFNUM=r
Identifies a register which (directly) contains the reference number of a SOLSIG entry.

(r)
Register containing the reference number.

Return information and error flags

During the execution of the macro, register R1 contains the reference number. Register R0 is overwritten with an internal function code.

R15:



A structured return code relating to the execution of the macro is transferred in register R15
(aa = primary return code, bb = secondary return code).

X'bb'	X'aa'	Meaning
X'00'	X'00'	Normal execution: the event has occurred. A post code has been transmitted, if it was specified
X'30'	X'00'	Function executed: the post code could not be transmitted because the receiver had not prepared a receiving field
X'34'	X'00'	Function executed: the post code was not transmitted because it had the value X' 00000000'
X'38'	X'00'	Function executed: post code truncated from the right (receiving field too small)
X'3C'	X'00'	Function executed: post code entered left-justified (receiving field too long). This return information does not occur if the RPOSTNUM operand is used in the DSOFEI macro.
X'04'	X'04'	No action: incorrect reference number (SOLSIG entry already deleted?)
X'20'	X'04'	No action: the event did not occur within the waiting time
X'28'	X'04'	No action: the event item was disabled (using the macro DISEI) before the event occurred
X'50'	X'04'	No event available.

SAVE – Save register contents

General

Application area:	Starting, interrupting and terminating; see page 72
Macro type:	Type O; see page 28

Macro description

The **SAVE** macro may be used to store register contents temporarily.

The **SAVE** macro is called at the beginning of a subroutine in order to save the register contents of the main program. A return to the main program is effected by the **RETRN** macro, which reloads the saved contents to the registers and makes the branch. All general registers except register 13 can be saved. Register 13 must contain the address of the save area, which must be defined by the main program.

Macro format and description of operands

SAVE
[(r1[,r2])] [,T] [,entry / *]

r1
Specifies a register (general register) which is to be saved. The numbers from 0 to 15 (except 13) may be specified for “r1“. The parentheses are optional.

r1,r2
Specifies a consecutive number of registers (general registers), which are to be saved. The range r1, r2 must not contain register 13, but but it may cross the register 15 / register 0 boundary.
Example: If (14,12) is specified, all registers except register 13, i.e. registers 0 to 12, 14 and 15, are saved.

Note

Register specifications can also be entered in the form “Rn”, where n = register number.

T
Is designed for compatibility with TOS language processors and specifies that registers 14 and 15 are to be saved. This operand is specified if “r1” or the range r1, r2 does not start or end on registers 14 and 15.

Note

> If the range r1, r2 starts with register 1 or 2 when T is specified, then all registers from 14 to “r2” are saved, i.e. registers 0 and possibly 1 are not excluded from the range.

entry
Provided for compatibility with TOS language processors and specifies an ID for the **SAVE** macro. “entry” must not comprise more than 155 characters; commas and blanks are not permissible. The ID is included in the macro expansion at a halfword boundary before the first instruction to be executed. The ID is immediately preceded by a byte beginning on a halfword boundary and containing the length of the ID.

*****
Provided for compatibility with TOS language processors and specifies that the entry in the name field of the macro is to be used as an ID. If the name field is empty, the name of the CSECT which contains the macro is used.

Functional description

The **SAVE** macro is used in subroutines, together with the **RETRN** macro, to save the contents of the main program's general registers. The **SAVE** macro stores the values, while the **RETRN** macro reloads them to the registers before the return branch and transfers a return code if desired.
The buffer for the register contents must be defined by the main program with a length of 18 words (72 bytes). Its address must be loaded to register 13.

Buffer structure

1	reserved for compiler	
2	address of the calling program's buffer (GR13)	reserved for program nesting
3	address of the called program's buffer	
4	general register 14, return address	saved register contents
5	general register 15, destination address	
6	general register 0	
7	general register 1, data addresses	
8	general register 2	
9	general register 3	
10	general register 4	
11	general register 5	
12	general register 6	
13	general register 7	
14	general register 8	
15	general register 9	
16	general register 10	
17	general register 11	
18	general register 12	

The buffer always contains the register contents of the program in which it is defined. The registers are stored by the called subroutine that calls the **SAVE** macro at the beginning.

One word of the buffer is allocated permanently to each of the registers 0 to 15 (except 13). If a register is not to be saved, the **SAVE** macro skips the associated buffer word. The order of the register contents in the buffer (14, 15, 0 to 12) permits all registers to be stored consecutively (excluding register 13). Register 13 cannot be saved by means of the **SAVE** macro, because it is used by **SAVE** for addressing the buffer.

If the subroutine changes register 13, this register must be saved separately. The **SAVE** macro cannot be used for this purpose.

Note on program nesting

A subroutine calling another subroutine, which also saves registers by means of **SAVE/RETRN**, must also define a buffer. Before it loads the address of this buffer to register 13, it must save the contents of register 13 (the address of the previous program's buffer) in word 2 of its own buffer. Before it returns to the calling program it must reload register 13 from that location.
It may deposit the address of its own buffer in word 3 of the buffer of the calling program.

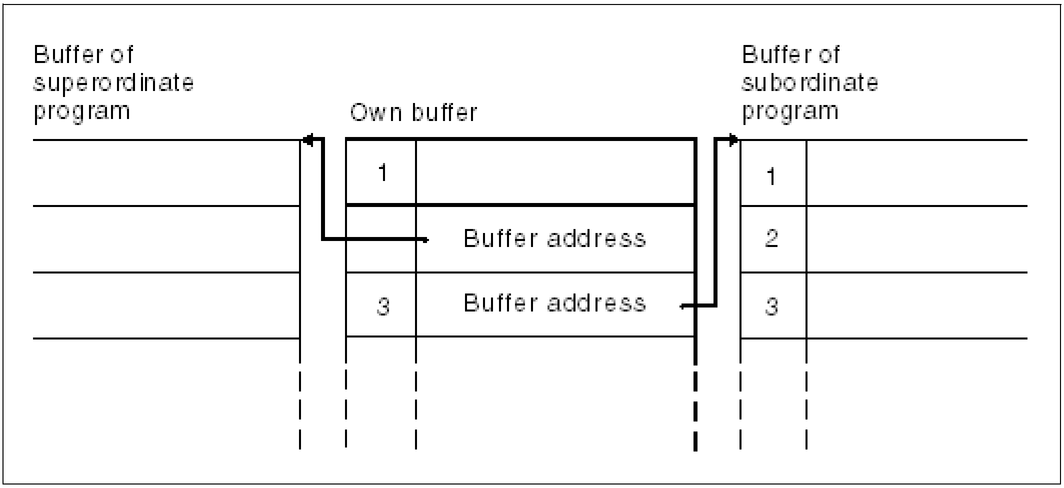

Buffer of superordinate program
Own buffer
Buffer of subordinate program
1
Buffer address
Buffer address
3
1
2
3

Figure 25: Buffer

The address in word 2 is entered by the program that defined the buffer.
The address in word 3 and the register contents in words 4 to 18 are entered by the called subroutine (**SAVE** macro).

Register conventions in nested programs:

Register R13	buffer address
Register R14	return address
Register R15	destination address (entry point in the called subroutine) or return code (see the **RETRN** macro)
Register R1	address of data addresses (if data addresses are to be transferred to the called subroutine).

SEGLD – Load segments

General

Application area: Linking and loading; see page 47
Macro type: Type O; see page 28

Macro description

The **SEGLD** macro permits automatic loading of a segment, even if it already resides in memory. Further segments within the same path of the overlay structure are loaded automatically.
A continuation address can be specified.

Macro format and description of operands

SEGLD
symbol1[,symbol2]

symbol1
Symbolic address within the segment to be loaded. A 4-byte V-type constant is generated for this symbol.

symbol2
Symbolic address in the calling or another module to which control is passed once loading has been completed (no V-type constant). If this operand is omitted, control is returned to the instruction following the **SEGLD** macro.

Functional description

The **SEGLD** macro (and also the **CALL** macro) causes the automatic loading of segments (nonautomatic loading is performed by the LPOV macro).
A statement in the **SEGLD** macro causes the Assembler to generate a V-type constant from the symbolic address specified. This constant identifies the segment to be loaded. Based on the V-type constant in the **SEGLD** macro (and on the CONTROL=YES operand in the PROGRAM control statement) the linkage editor generates an overlay control module required for automatic loading which satisfies the V-type constant with an address. When the **SEGLD** macro is executed, control is passed to the overlay control module. The segment that contains the symbolic address as well as all further segments within the same path of the overlay structure are then brought into memory by the overlay control module. This takes place regardless of whether or not the segments are already in memory.

Upon completion of the loading process control is returned to the instruction following the **SEGLD** macro (if the “symbol2” operand is omitted). The optional address provided by the “symbol2” operand can be present within the calling module or may be an external reference. In the latter case the user must issue the corresponding ENTRY and EXTRN statements as well as making sure that the module that contains “symbol2” is in memory after completion of the loading process (see the “Utility Routines” manual [27]).

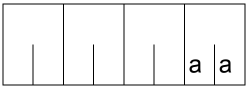
Notes on the macro call

- The overlay structure of a program in which automatic loading of segments is performed should be designed in such a way as to allow for additional memory requirements for the overlay control module, for ENTAB and for SEGTAB (see the “Utility Routines” manual [27]).
- The **SEGLD** macro must be contained in a program area that is covered by a USING statement. As register 15 is used by the **SEGLD** macro for segment loading, it must not be used as the base register for the program section that contains the **SEGLD** macro.

Return information and error flags

If an error occurs while an overlay segment is being loaded, the Executive continues the program with the command following **SEGLD**. If no error occurs, the continuation address specified in the macro applies.

R15:



A return code relating to the execution of the SEGLD macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Segment has been loaded
X'04'	Incorrect read-only modification record
X'08'	Incorrect record code in text/modification block, or there is a modification record before the 1st text record
X'0C'	Too many read-only modification records present, or segment name cannot be found in any of the index records for load modules (= segment)
X'10'	There is no index record for load modules with segment name %ROOT
X'14'	There is insufficient memory space to load the segment
X'18'	Incorrect code information, or error in reading PAM load module file, or error in message output
X'1C'	Incorrect segment name
X'20'	Error in loading AID
X'24'	Error in reading a C element of a PLAM library

SELPRGV – Select program version

General

Application area:	Linking and loading; see page 47
Macro type:	Type S, MF format **2**: standard /C/D/E/L/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **SELPRGV** macro determines which program version the DBL is to use if several versions of a program can be loaded. This program must not been loaded when the version is selected.

Macro format and description of operands

SELPRGV
MF=S / C / D / E / L / M ,PRGNAME=<name 1..32> ,PRGNAM@=<var:name 32..32> / (<reg: pointer>) ,PRGVERS=<name 1..24> / *STD ,PRGVER@=<var:name 24..24> / (<reg: pointer>) ,SCOPE=PROGRAM / TASK ,PARAM=<var: pointer> / (reg: pointer>) ,PREFIX=P / p ,MACID=BSL / macid

PRGNAME = <name 1..32>
Program name. As far as the DBL is concerned this is the name of a load unit. The name may contain alphanumerical characters only. May be specified only with MF=L or MF=S.

PRGNAM@ = <var: name 32..32> / (<reg: pointer>)
Symbolic address or register containing the address of a 32 character field, which contains the the program name. Shorter name specifications must be padded with blanks.
May be specified only if MF=M.

PRGVERS = <name 1..24>
Program version to be used by the DBL. May be specified only with MF=L or MF=S.

***STD**
No version is selected. The DBL deletes the program version from its version table.

PRGVER@ = <var: name 24..24> / (<reg: pointer>)
Symbolic address or register containing the address of a 32 character field, which contains the the program version. Shorter version specifications must be padded with blanks.
May be specified only if MF=M.

SCOPE=
Scope for the version selection.

PROGRAM
Default setting: The version selection only remains valid until the program is terminated or the version selection is deleted. Version selection must therefore be repeated before or during every program call.

TASK
The version selection remains valid until the end of the task or until the version selection is deleted.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM, PREFIX and MACID, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

It is possible to specify a PREFIX in the C form, D form, or M form of the macro, and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

Notes on the macro call

- Valid class 6 memory addresses must be specified in PRGNAM@ and PRGVER@.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the SELPRGV macro is transferred in the standard header
(cc=subcode2,bb=subcode1,aaaa=main code)

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally
X'00'	X'01'	X'0001'	PRGNAME not specified or invalid
X'00'	X'01'	X'0002'	PRGVERSnot specified or invalid
X'00'	X'01'	X'0003'	Invalid specification in SCOPE
X'00'	X'00'	X'0004'	Version selection cannot be deleted since no program version has been selected
X'00'	X'01'	X'0005'	Program version table cannot be generated. Version selection rejected
X'00'	X'20'	X'0006'	DSSM error
X'00'	X'20'	X'0300'	System error
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported
X'00'	X'03'	X'FFFF'	The interface version is not supported

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

SETBF – Set buffer size for dialog communication

General

Application area:	Data terminal communication; see page 160
Macro type:	Type O; see page 28

Macro description

The **SETBF** macro enables a user program to change the size of the internal physical I/O buffer for communication with the terminal.
The **SETBF** macro is ignored in batch mode.

Macro format and description of operands

SETBF
$\left\{\begin{array}{l}\text{size [,N]}\\ \text{(1)}\end{array}\right\}$

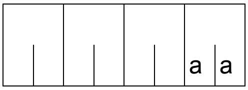
size
Size of the buffer (number of characters). 80 ≤ size ≤ 3482.

N
Indicates that no change is to be made in the buffer size if the requested size is the same or smaller than the current buffer size.

(1)
Indicates that the user has loaded general register 1 with the size of the buffer desired before issuing the **SETBF** macro. If the function corresponding to the N operand is required, the complement of the size must be loaded in register R1.

Return information and error flags

R15:



A return code relating to the execution of the SETBF macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Requested buffer allocated
X'04'	Invalid buffer size specified by the user program
X'08'	New buffer could not be allocated

SETIC – Set interval timer

General

Application areas:	Starting, interrupting and terminating; see page 72 STXIT processing; see page 131
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

Macro description

The **SETIC** macro enables a time interval for the CPU time and/or the real time to be defined and the summer/winter time change event to be indicated. After the time interval has elapsed, a “CPU time interval elapsed” or “real time interval elapsed” or “summer/winter time change” interrupt event is signaled, and a STXIT routine allocated in the calling program is activated (see the **STXIT** macro). If this does not happen, the program is terminated.

Functional description

The Executive sets interval timers to the values specified in the **SETIC** macro. As soon as these values are reached, the user program is interrupted. The system generates the event X'20' for “CPU time interval elapsed” and/or X'A0' for “Real time interval elapsed” and/or X'20' for “summer/winter time change”, and - if specified by **STXIT** - control is passed to the interrupt routine for the appropriate timers. If no interrupt routine was specified in the **STXIT** macro, the program is terminated on occurrence of a timer interrupt.
Once the interrupt has been initiated, the Executive resets the timer to the value specified in the **SETIC** macro. The interrupts will occur regularly at given intervals or times of day, as specified, until the the time interval is changed by another **SETIC** macro or deactivated by specification of the value zero. If “REPEAT=NO” is specified, repeating of the intervals can be suppressed.
If the task is in a **PASS**/**VPASS** wait state when the real-time interval elapses, this wait state is not interrupted. Control is not passed to the specified **STXIT** routine until the wait state has terminated.

Macro format and description of operands

SETIC
[CPUTIM=addr / (r)] [{ ,REALTIM=addr / (r) }]   { ,TOD=addr / (r) } [,CHWSTIM=YES / NO] ,REPEAT=<u>YES</u> / NO [,PARMOD=24 / 31] [,MF=L / (E,..)]

CPUTIM=
Makes specifications regarding the CPU time interval (event code X'20').

addr
Symbolic address of the field that contains the CPU time interval specification. There are two possible field formats:

- The “addr” field is one word long and contains the time interval as a binary number whose value is interpreted in terms of milliseconds.
- The “addr” field is 6 bytes long and contains the time interval in the format hhmmss - hours, minutes seconds (EBCDIC).

Maximum value for hours: 24
Maximum value for minutes and seconds: 59
Once the specified time has elapsed, control is passed to the routine specified by the **STXIT** macro (TIMER operand).

(r)
Register containing the address value of the “addr” field.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

REALTIM=
Makes specifications regarding the real-time interval (event code X'A0').

addr
Symbolic address of the field that contains the real-time interval. There are two possible field formats:

– The “addr” field is one word long and contains the real-time interval as a binary number whose value is interpreted in terms of milliseconds.
– The “addr” field is 6 bytes long and contains the real-time interval in the format hhmmss - hours, minutes, seconds (EBCDIC).

Maximum value for hours: 24
Maximum value for minutes and seconds: 59
The value 0 is interpreted as 24 hours.
Once the specified time has elapsed, control is passed to the routine specified in the **STXIT** macro (RTIMER operand).

(r)
Register containing the address value of the “addr” field.

REPEAT=
Specifies whether or not the same interval is to be set again after it has elapsed.

YES
Default setting: the same interval is to be set again after it has elapsed. This operand is effective only in conjunction with the “CPUTIM”, “REALTIM” or “TOD” operand. With “TOD”, it will be repeated every 24 hours. With REALTIM all values < 50 msec are set to 50 msec. The summer/winter time change (or vice versa) is always implemented with REPEAT=YES.

NO
The interval will not be set again.

TOD=
Makes time-of-day specifications for a real timer, based on a 24-hour clock (event code X'A0').
The specification is made in the format hhmmss - hours, minutes, seconds (EBCDIC).
When the specified time of day is reached, control is passed to the routine specified in the **STXIT** macro (RTIMER operand).

addr
Symbolic address of a 6-byte field containing the time-of-day specification.

(r)
Register containing the address value of the “addr” field.

CHWSTIM=
Indicates the summer/winter (or vice versa) time change event (event code X'C0'). When the event occurs, control is transferred to the routine specified in the **STXIT** macro (RTIMER operand).

YES
The event is indicated, i.e. the STXIT routine - if defined - is started.

NO
The event is not indicated.

Return information and error flags

R15:

						a	a

A return code relating to the execution of the SETIC macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Normal execution
X'04'	Function was not executed. Invalid operands
X'08'	Function was not executed. Invalid time entry

SEVNT – Send event

General

Application areas:	Intertask communication; see page 76 Communication; see page 163
Macro type:	Type O; see page 28

Macro description

A user participating in intertask communication may transfer a message to another ITC participant by means of the **SEVNT** macro.

Macro format and description of operands

SEVNT
sender-field,receiver-name (1)

sender-field
Symbolic name of the field containing the message to be transferred. This field must start at a word boundary and must be constructed like a variable-length record (with 4 byte record length field):

Bytes 0-1: record length in bytes = length of message +4
(8 ≤ record length ≤ 65 535)
Bytes 2-3: reserved
Bytes 4-n: message

The message (including the record length field) must be at least 8 bytes long and must not exceed 65535 bytes.

receiver-name
ITC name of the participant which is to receive the message.

(1)
Specifies that register R1 contains the address of an operand field. The operand field must be aligned on a word boundary and have the following contents:

Byte 0-3: Address of the sender field (format as described above).
Byte 4-11: ITC name of the receiver. If the name is shorter than 8 characters, the field must be padded with blanks (X'40').

Functional description

Each ITC participant may transfer a message to another participant by means of the **SEVNT** macro. The message may be between 4 bytes and 64 Kbytes long. It is transferred into the receive queue of the receiving ITC participant. The **SEVNT** macro is rejected if there is not enough system memory available, or if the total length of messages in the receive queue exceeds a predefined limit (128 K). The sending program is not automatically informed whether the receiver requests and analyzes the message from the receive queue. To obtain this information, the receiver would have to return a message as acknowledgment. The task which includes the **SEVNT** call is not interrupted by the sending of the message.

An ITC participant may issue successive **SEVNT** calls to send messages to various receivers.

Return information and error flags

R15:

						a	a

A return code relating to the execution of the SEVNT macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	The message was transferred to the receiving queue of the specified participant
X'04'	Operand error (e.g. memory is not allocated or is not class 6 memory). The message was not transferred
X'08'	The calling task is not an ITC participant. The message was not transferred
X'0C'	Not enough system memory available or the internal limit for receive queues has already been exceeded. The message was not transferred
X'10'	The receiver is not (yet) an ITC participant, or the calling task is trying to transfer a message to itself. The message was not transferred

SOLSIG – Solicit signal request

General

Application area:	Eventing; see page 94
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

If the 24-bit interface is used, a 4-byte field is generated for the post code. If the 31-bit interface is used, the post code may have a length of either 4 or 8 bytes.

Macro description

This macro is used to issue a “SOLicit SIGnal request” for an event item. The event item must previously have been assigned to the calling task (by **ENAEI**). The task can wait until an event occurs (synchronous operation) or, if the task is to be continued, until a contingency process is initiated as a result of the event (asynchronous operation).
In either case the task is continued (or the contingency process is initiated) once a specified period of time has elapsed - even if the event does not occur.
The contingency process must previously have been defined (by **ENACO**).

Note

> If a program (package) has defined a contingency process written in SPL, register R12 must contain the address of the SPL program manager for **all ENACO**, **SOLSIG** and **POSSIG** calls.

Macro format and description of operands

```
SOLSIG
-----------------------------------------------------------------------------------------------
{ { EINAME=name                                 }                                              }
{ {                                             }                                              }
{ { EINAMAD={ addr }[,EINAMLN=length]           }[,SCOPE=LOCAL / GROUP / USER_GROUP / GLOBAL]  }
{ {         { (r)  }                            }                                              }
{                                                                                              }
{ EIID={ addr }                                                                                }
{      { (r)  }                                                                                }
```

```
SOLSIG (cont.)
-------------------------------------------------------------------------------------------

 {                                                                                       }
 {                  { RPOSTAD={ addr }  }                                                }
 {  ,COND=UNCOND[,{           { (r)  }  }][,RPOSTL={ 1 }][,LIFETIM={ sec }]              }
 {                  {                    }           { 2 }            { (r) }              }
 {                  { RPOSTR=r           }                                                 }
 {                                                                                       }
 {                { RPOSTAD={ addr }  }                                                  }
 {  ,COND=IMMED[,{          { (r)  }  }] ,RPOSTL={ 1 }                                   }
 {                {                    }          { 2 }                                    }
 {                { RPOSTR=r           }                                                   }
 {                                                                                       }
 {  ,COID={ addr }[,COMAD={ addr }][,COND=PERM][,LIFETIM={ sec }]                        }
 {        { (r)  }        { (r)  }                       { (r) }                           }

[,PARMOD=24 / 31]
[,MF=L / (E,..)]
```

EINAME=name
Specifies the name of the event item to which the requested event is signaled. The event item must already have been defined (**ENAEI**). The name of the event item is unique only in conjunction with SCOPE.

EINAMAD=
Specifies the name of the event item. This entry is unique only in conjunction with SCOPE.

addr
Symbolic address of the field containing the name.

(r)
Register containing the address of the field.

EINAMLN=length
Specifies the length in bytes of the event item name. If the operand is missing, the length attribute of the EINAMAD operand is assumed if EINAMAD=addr is specified;
if EINAMAD=(r), the maximum length (54 bytes) is assumed.

length
Length of the event item name.

SCOPE=
Specifies the scope (group of participants) of the event item.

LOCAL
Default setting: the use of the event item is limited to the calling task.

GROUP
All the tasks with the same user ID as the calling task are participants.

USER_GROUP
All the tasks, whose user IDs belong to the same user group as the user ID of the creating participant, can be participants.
The operand value assumes the existence of user groups and may therefore only be specified when the SRPM function unit of the SECOS software product is available in the system. This is why the GETUGR macro (see the “SECOS” manual [14]) has to check whether SRPM is available prior to a macro call with SCOPE=USER_GROUP. The program reaction is dependent on the result (return code).

GLOBAL
All the tasks in the system are participants.

EIID=
Specifies the ID of the event item. This ID is supplied to the user by the **ENAEI** macro. If the ID is used instead of the name of the event item, processing is speeded up. The ID is unique.

addr
Symbolic address of a 4-byte field containing the ID.

(r)
Register containing the address of the field.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

Operands for synchronous operation

COND=
Specifies whether or not the caller is willing to wait for the event to occur.

UNCOND
Denotes that the caller is willing to wait for the event to occur. The waiting time can be restricted with the aid of the LIFETIM operand.

IMMED
Denotes that the caller is not willing to wait for the event to occur. The program must be continued even if the event was not yet signaled to the event item by a **POSSIG** macro. More information is supplied by the secondary indicator X'20' in register 15 (see “Return information and error flags” below).

RPOSTAD=
Specifies a field to which a post code is to be transferred. The post code may be 4 or 8 bytes long. The RPOSTL operand determines whether only the first word or both words of the post code are to be placed in the field.
The RPOSTR operand has the same function as this operand and, if it is used, processing is speeded up.
The address value 0 is not allowed.

addr
Symbolic address of the field to receive the post code. Field length = 4 or 8 bytes.

(r)
Register containing the address value “addr”.

RPOSTR=r
Indicates a register where the post code is to be entered directly. A two-word post code is entered in this register and the one following it (in number) if RPOSTL=2 is specified.

r
Register to receive the post code.

RPOSTL=
Gives the length in words of the post code to be received. If the 24-bit interface (PARMOD=24) is used, only RPOSTL=1 is permitted.

1
Default setting: only one word (the first word) of the post code is to be transferred.

2
The complete post code (2 words) is to be transferred.

LIFETIM=
Time for which the task is to wait for an event to occur. The return information indicates whether the request was satisfied or the waiting time elapsed. The operand is ignored if COND=IMMED is specified.

sec
Time in seconds. 1 ≤ sec ≤ 43200
The processing accuracy is +10 seconds.
Default value: 600 sec.

(r)
Register containing the specification in seconds.

Operands for asynchronous operation

COID=
Specifies the ID of the contingency process. The ID is supplied to the user by the **ENACO** macro call.

addr
Symbolic address of a 4-byte field containing the ID.

(r)
Register containing the address.

COMAD=
Specifies a contingency message. A contingency message issued here replaces any which might have been issued when the contingency was defined (by **ENACO**).

addr
Symbolic address of a word containing a contingency message.

(r)
Register containing the address.

COND=PERM
Permanent asynchronous **SOLSIG**:
If the signal requested by **SOLSIG** arrives (via **POSSIG**) within the waiting time (LIFETIM), another **SOLSIG** macro is issued. The waiting time for this second macro is 600 seconds. If no signal arrives during this waiting time, no further **SOLSIG** macro is issued.

LIFETIM=
Time during which the event should occur. The event information code notifies the contingency task as to whether or not the request was honored within the specified period of time.

sec
Time in seconds. 1 ≤ sec ≤ 43200
The processing accuracy is +10 seconds.
Default value: 600 sec

(r)
Register containing the time specification in seconds.

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:

b	b					a	a

A structured return code (aa = primary return code, bb = secondary return code) relating to the execution of the macro is transferred in register R15.

X'bb'	X'aa'	Meaning
X'00'	X'00'	Function executed: – synchronous operation: The event occurred. Post code information was provided and received, or post code information was neither provided nor requested. – asynchronous operation: SOLSIG was successfully queued
X'30'	X'00'	Function executed: post code information was provided (POSSIG) but SOLSIG did not request a destination field (RPOSTAD)
X'34'	X'00'	Function executed: no post code information was provided (POSSIG) although SOLSIG requested such information (RPOSTAD). A zero post code (X'000000') is regarded as no post code
X'38'	X'00'	Function executed: post code exceeds specified receiving field in length. The 2nd word (4 rightmost bytes) has been truncated
X'3C'	X'00'	Function executed: post code is shorter than the receiving field. It is entered left-justified
X'0C'	X'04'	No action: the event item established by the system has not been assigned to the calling task
X'10'	X'04'	No action: invalid operands were specified
X'14'	X'04'	No action: invalid name or ID. No event item with the specified identification exists
X'18'	X'04'	No action: maximum number (400) of contingency processes allowed per basic task has been exceeded
X'20'	X'04'	No action: the event did not occur. Either COND=IMMED was set before POSSIG had been issued or the waiting time has elapsed
X'24'	X'04'	No action: invalid contingency process ID. No contingency process with this ID exists
X'28'	X'04'	No action: the event item was deleted before the event occurred

Example: synchronous operation

```
  SOLSIG   START
           PRINT NOGEN
  SOLSIG   AMODE ANY
           GPARMOD 31
1                  *,MACRO: GPARMOD, VERSION: VER121
           BALR  3,0
           USING *,3
           ENAEI EINAME=EVENT,SCOPE=GLOBAL,EIIDRET=KK ─────────────────  (1)
           GDATE TOD=TIME1
           SOLSIG EIID=KK,COND=UNCOND ─────────────────────────────────  (2)
  *** WAITING FOR SIGNAL ***
           GDATE TOD=TIME2
           ST    15,RCFIELD1
           GDATE TOD=TIME3
           SOLSIG EIID=KK,COND=UNCOND,LIFETIM=70 ──────────────────────  (3)
  *** WAITING FOR SIGNAL ***
           GDATE TOD=TIME4
           ST    15,RCFIELD2
           GDATE TOD=TIME5
           SOLSIG EIID=KK,COND=IMMED ──────────────────────────────────  (4)
           GDATE TOD=TIME6
           ST    15,RCFIELD3
           CHKEI EIID=KK ──────────────────────────────────────────────  (5)
           ST    15,RCFIELD4
           DISEI EIID=KK
  DTH1     TERM
  ****   DEFINITIONS   *****
  KK       DS    F
  TIME1    DS    CL8
  TIME2    DS    CL8
  TIME3    DS    CL8
  TIME4    DS    CL8
  TIME5    DS    CL8
  TIME6    DS    CL8
  RCFIELD1 DS    F
  RCFIELD2 DS    F
  RCFIELD3 DS    F
  RCFIELD4 DS    F
           END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,solsig), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,solsig)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 453 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 84 MSEC
//end
%  ASS6012 END OF ASSEMBH
/load-executable-program library=macexmp.lib,element-or-symbol=solsig, -
/     test-options=*aid
%  BLS0523 ELEMENT 'SOLSIG', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'SOLSIG', VERSION ' ' OF '<date> <time>' LOADED
/%in dth1;%r
STOPPED AT LABEL: DTH1 , SRC_REF: 287, SOURCE: SOLSIG , PROC: SOLSIG
/%d %@(time1) -> %cl8,%@(time2) -> %cl8 ───────────────────────────────  (6)
*** TID: 005000D8 *** TSN: 2QSE ******************************************
**
CURRENT PC: 00000136    CSECT: SOLSIG  **************************************
**
V'00000158' = SOLSIG   + #'00000158'
00000158 (00000158) 13:14:15
V'00000160' = SOLSIG   + #'00000160'
00000160 (00000160) 13:24:53
/%d %@(rcfield1) -> %x ──────────────────────────────────────────────  (7)
V'00000188' = SOLSIG   + #'00000188'
00000188 (00000188) 20000004                                   ....
/%d %@(time3) -> %cl8,%@(time4) -> %cl8 ───────────────────────────────  (8)
V'00000168' = SOLSIG   + #'00000168'
00000168 (00000168) 13:24:53
V'00000170' = SOLSIG   + #'00000170'
00000170 (00000170) 13:26:02
/%d %@(rcfield2) -> %x ──────────────────────────────────────────────  (9)
V'0000018C' = SOLSIG   + #'0000018C'
0000018C (0000018C) 20000004                                   ....
/%d %@(time5) -> %cl8,%@(time6) -> %cl8 ─────────────────────────────── (10)
V'00000178' = SOLSIG   + #'00000178'
00000178 (00000178) 13:26:02
V'00000180' = SOLSIG   + #'00000180'
00000180 (00000180) 13:26:02
```

```
/%d %@(rcfield3) -> %x  ─────────────────────────────────────────────── (11)
V'00000190' = SOLSIG   + #'00000190'
00000190 (00000190) 20000004                               ....
/%d %@(rcfield4) -> %x  ─────────────────────────────────────────────── (12)
V'00000194' = SOLSIG   + #'00000194'
00000194 (00000194) 30000000                               ....
/%r
```

(1) The event item EVENT is defined. KK is the address for the ID.

(2) The task solicits a signal from the event item. The caller is willing to wait for the arrival of the signal until the end of the default waiting time (10 minutes).

(3) A second signal is requested. In this case, the waiting time is only 70 seconds.

(4) A third signal is requested. The task will not wait for it. As no signal has been posted in the POSSIG event queue, the task is continued immediately.

(5) The queues of the event item EVENT are checked. They are empty because the waiting times of all **SOLSIG** macros have expired. Then the definition of the event item is disabled.

(6) Time of day before and after the **SOLSIG** macro with default waiting time: the waiting time was 10 minutes and 38 seconds.

(7) Return code X'20000004' after the first **SOLSIG** call: the event did not take place.

(8) Time of day before and after the **SOLSIG** call with 70 seconds waiting time: the waiting time was 69 seconds.

(9) Return code X'20000004' after the second **SOLSIG** call: the event did not take place.

(10) Time of day before and after the **SOLSIG** macro without waiting time: no waiting time.

(11) Return code X'20000004' after the third **SOLSIG** call: the event did not take place.

(12) Result of the check of the receive queues by means of **CHKEI**: there are no requests in the receive queues.

For **examples**, see the sections “Eventing” (page 106) and “Contingency processes” (page 118).

SRMUINF – Read user information from user catalog

General

Application area: Requesting and accessing lists and tables; see page 155
Macro type: Type S, MF format **3**: D/C/E/L/M form; see page 29

An entry is made by a user administrator in the user catalog for each user (user ID). The contents of the entry include the following:

- user ID, password authorization,...
- specifications on system resources available to the user (CPU time, memory space,...)
- special user rights (privileged access,...)
- accounting data.
- Mailing information (mail and email addresses)

Macro description

The **SRMUINF** macro reads data from the user catalog and transfers it to an area specified beforehand. Depending on the specification made, the accounting data, the user-specific data or the entire entry for a user ID is output from the user catalog. The following distinction needs to be made:

- Macro under the ID of the nonprivileged user:
  only data from the user's own entry is output.
- Macro under the ID of a user administrator with the system privilege “USER-ADMINISTRATION” (normally the user ID TSOS or, if the software product SECOS is used, a user ID with the corresponding privileges):
  - output of data from the user's own entry
  - output of data from other users (individually for any one user or in sequence for more than one user)
- Macro under the ID of a user group administrator (only possible if the software product SECOS is used). Only data from a local pubset (and not a shared pubset) can be output:
  - output of data from the user's own entry
  - output of data from other users who belong to the group or subgroup of the user group administrator.

Macro format and description of operands

```
SRMUINF

INFO=*USER / *ALL / *ACCOUNT / *POSIX / *EMAIL / *ALL_EMAIL / addr / (r)
[,AREA@=addr / (r)]
[,AREA#=length / addr / (r)]
,USERID=*OWN / *FIRST / 'userid' / addr / (r)
,PVS=*HOME / 'catid' / addr / (r)
,ACTION=*READ / *READNXT / *READSEQ / addr / (r)
          ⎧ PARAM                                                     ⎫
,XPAND=   ⎨                                                           ⎬
          ⎩ OUTPUT,DATA=ALL / USER / ACCOUNT / POSIX / EMAIL / ALL_EMAIL ⎭
,MF=D / C / L / M / E
[,PARAM=addr / (r)]
,PREFIX=S / p
,MACID=RMV / macid
```

The operands are described below in alphabetical order.

ACTION=
Specifies whether data is to be read from the user catalog for the specified user ID or the next user ID.

***READ**
Default setting: the entry for the specified user ID is read from the user catalog.

***READNXT**
The entry for the user ID that follows the specified user ID is read from the user catalog.
This value can only be specified under the ID of a user administrator with the system privilege “USER-ADMINISTRATION” (this system privilege is normally assigned to the user ID TSOS or, if the software product SECOS is used, to a user ID with the corresponding privileges).

***READSEQ**
The entry for the user ID that follows the specified user ID is read from the user catalog. In addition, the user ID whose entry was read is entered in the parameter list as the USERID operand.

addr
Symbolic address (name) of the field containing the information as to which entry from the user catalog is to be transferred.
May be specified only in conjunction with MF=M.

(r)
Register containing the address value for the value of the ACTION operand. May be specified only in conjunction with MF=M.

AREA@=
Denotes the address of an output area to which the data from the user catalog is transferred.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value of the output area. May be specified only in conjunction with MF=M.

AREA#=
Specifies the length of the output area. The minimum length depends on the amount of information to be output (determined by means of the INFO operand). The entry is truncated if the length specification is inadequate (return code aa = X'10').

length
Integer specifying the length of the output area in bytes. “length” can assume the values 0, 1, ..., 4096.

addr
Symbolic address (name) of a field containing the length of the output area. May be specified only in conjunction with MF=M.

(r)
Register containing the address value for the length of the output area. May be specified only in conjunction with MF=M.

DATA=
Controls the definition of specific output areas. This operand is evaluated only if XPAND=OUTPUT is specified.

ALL
Default setting: defines the complete entry for the specified user ID, but without the EMAIL- and POSIX-specific parts.

USER
Defines the user-specific part (without account numbers) of the entry for the specified user ID.

ACCOUNT
Defines the account-specific part (account numbers only) of the entry for the specified user ID.

POSIX
Defines the POSIX-specific part of the entry for the specified user ID.

EMAIL
Defines the EMAIL-specific part of the entry for the specified user ID.

ALL_EMAIL
Defines the complete entry with the EMAIL-specific part for the specified user ID, but without the POSIX-specific part.

INFO=
Specifies how much data is to be transferred from the entry in the user catalog.

***USER**
Default setting: only the user-specific data is transferred.

***ALL**
The entire entry is transferred, but without the EMAIL- and POSIX-specific parts.

***ACCOUNT**
Only the account-specific data is transferred.

***POSIX**
The POSIX-specific data is transferred. The same data is output with the SHOW-POSIX-USER-ATTRIBUTES command.

***EMAIL**
The area with the receiver addresses for emails is transferred.

***ALL_EMAIL**
The complete entry and the area with the receiver addresses for emails is transferred, but not the POSIX-specific part.

addr
Symbolic address (name) of a field containing the information as to how much data is to be transferred. May be specified only in conjunction with MF=M.

(r)
Register containing the address value for the value of the INFO operand. May be specified only in conjunction with MF=M.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29.
The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

PVS=
Specifies the PVS (**P**ublic **V**olume **S**et; “pubset”) of the user catalog from which an entry is to be read.

*HOME
Default setting: the desired entry is contained in the user catalog of the home pubset.

'catid'
Catalog ID of the PVS containing the user catalog from which an entry is to be read.
'catid' has a maximum length of 4 characters.

addr
Symbolic address (name) of a 4-byte field containing the 'catid'. Short catids have to be padded with blanks.
May be specified only in conjunction with MF=M.

(r)
Register containing the address of the 4-byte 'catid'. Short catids have to be padded with blanks.

USERID=
Denotes the user ID whose entry is to be read from the specified user catalog.

*OWN
Default setting: the entry for the caller's own user ID is output.

***FIRST**
The first entry in the specified user catalog is output.
This value may be specified only in conjunction with the ACTION=*READNXT operand and only if the caller has the appropriate privileges.

'userid'
User ID.
'userid' has a maximum length of 8 characters.
The privileges of the caller govern which user IDs may be specified. The following cases apply:

– The caller is a nonprivileged user:
  only the user's own ID is permissible.
– The caller is a user group administrator (only possible when the software product SECOS is used):
  any user ID from the caller's user group or from a subgroup of the caller's group is permissible.
– The caller is a user administrator with the system privilege “USER-ADMINISTRATION” (normally the user ID TSOS or, if the software product SECOS is used, a user ID with the corresponding privileges):
  any user ID on the specified PVS is permissible.

addr
Symbolic address (name) of an 8-byte field containing the 'userid'. May be specified only in conjunction with MF=M.

(r)
Register containing the address value “addr”. May be specified only in conjunction with MF=M.

XPAND=
Specifies which data area is to be processed.

<u>**PARAM**</u>
Default setting: the call data area is processed.

OUTPUT
Requests output of a special output area. The DATA operand must be specified. This determines the output layout of the desired information.

Return information and error flags

Standard header:

		b	b			a	a

A structured return code relating to the execution of the SRMUINF macro is transferred in the standard header: (aa = primary return code, bb = secondary return code)

X'bb'	X'aa'	Meaning
X'00'	X'00'	Normal execution
X'01'	X'04'	Operand error or insufficient privileges
X'00'	X'08'	No entry exists for this ID
X'80'	X'0C'	The pubset (PVS) is inaccessible
X'00'	X'10'	The entry for the user ID was not output in its entirety (inadequate length specification)

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Layout of the DSECT (1)

```
           SRMUINF MF=D,PREFIX=A
1          STACK  PRINT
1          PRINT  NOGEN
2                *,##### PREFIX=A, MACID=RMV #####
1          #INTF REFTYPE=REQUEST,INTNAME=SRMUINF,INTCOMP=001
1 ARMVPL     DS    0F          BEGIN of PARAMETERAREA
1          FHDR  MF=(C,ARMV),EQUATES=NO
2          DS    0A
2 ARMVFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 ARMVIFID DS    0A              0   INTERFACE IDENTIFIER
2 ARMVFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 ARMVFCT  DS    AL1             2   FUNCTION NUMBER
2 ARMVFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 ARMVRET  DS    0A              4   GENERAL RETURN CODE
2 ARMVSRET DS    0AL2            4   SUB RETURN CODE
2 ARMVSR2  DS    AL1             4   SUB RETURN CODE 2
2 ARMVSR1  DS    AL1             5   SUB RETURN CODE 1
2 ARMVMRET DS    0AL2            6   MAIN RETURN CODE
2 ARMVMR2  DS    AL1             6   MAIN RETURN CODE 2
2 ARMVMR1  DS    AL1             7   MAIN RETURN CODE 1
2 ARMVFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1 *                                       ADDITIONAL MAIN RETURN-CODES:
1 *
1 ARMVMOK  EQU   0                        REQUEST SERVICED
1 ARMVPAER EQU   4                        PARAMETER ERROR
1 ARMVENFO EQU   8                        ENTRY NOT FOUND
1 ARMVPNAC EQU   12                       PVS NOT ACCESSIBLE
1 ARMVRSER EQU   16                       RESOURCE NOT AVAILABLE
1 *
1          DS    XL4                      UNUSED
1 ARMVUSRD DS    CL8                      USERID SPECIFIED
1 ARMVINFO DS    AL1                      INFO-TYPE
1 ARMVIALL EQU   1                          ALL REQUIRED
1 ARMVIUSR EQU   2                          USER INFO REQUIRED
1 ARMVIACI EQU   3                          ACCOUNT INFO REQUIRED
1 ARMVIPOS EQU   4                          POSIX INFO REQUIRED
1 ARMVIEMA EQU   5                          EMAIL INFO REQUIRED
1 ARMVIAEM EQU   6                          ALL&EMAIL INFO REQUIRED
1 ARMVACOD DS    AL1                      ACTION CODE
```

```
1 ARMVACRD EQU   1                          READ
1 ARMVACRN EQU   2                          READ NEXT
1 ARMVPVS  DS    CL4                      PVS CATID
1 ARMVHOME EQU   C'#'                     PVS IS HOME
1          DS    CL4                      FOR CAT-ID EXTENSION
1          DS    XL2                      UNUSED1
1 ARMVARE@ DS    A                        AREA ADDRESS
1 ARMVARE# DS    AL2                      AREA LENGTH
1          DS    XL2                      UNUSED2
1 *
1 ARMVPL#  EQU   *-ARMVPL                 P/L LENGTH
```

Layout of the DSECT (2)

```
           SRMUINF MF=D,XPAND=OUTPUT,DATA=ALL_EMAIL
1          STACK  PRINT
1          PRINT  NOGEN
2                *,##### PREFIX=S, MACID=RMV #####
1 **************************************************************
1 *        INFORMATION ON JOB                                  *
1 **************************************************************
1 SRMVJOB  DS    0F
1 SRMVUSER DS    CL8                     USER-IDENTIFICATION
1 SRMVSVR  DS    CL1                     SEVER INDICATOR
1 SRMVSVRY EQU   1                       - USERID HAS BEEN SEVERED
1 SRMVSVRN EQU   2                       - USERID NOT      SEVERED
1 SRMVPVG  DS    CL1                     USER PRIVILEGE CODE
1 SRMVPVGA EQU   1                       - SYSTEM ADMINISTRATOR
1 SRMVPVGN EQU   2                       - NORMAL USER
1 SRMVPASS DS    CL8                     USER'S PASSWORD
1 SRMVENCR DS    CL1                     PASSWORD-ENCRYPTION
1 SRMVENUN EQU   0                       - UNDEFINED
1 SRMVENNO EQU   1                       - NO ENCRYPTION
1 SRMVESCA EQU   2                       - ENCRYPT-MODE SCA
1 SRMVEOLD EQU   3                       - ENCRYPT-MODE OLD
1 SRMVENOP EQU   4                       - NO PASSWORD
1          DS    XL1                     NOT USED
1 SRMVJOB#  EQU   *-SRMVJOB                 LENGTH OF JOB PART
1 **************************************************************
1 *        INFORMATION ON DMS                                  *
1 **************************************************************
1 SRMVDMSD DS    0F
1 SRMVSPLI DS    F                       SPACE LIMIT
1 SRMVSPUS DS    F                       SPACE USED
1          DS    XL2                     RESERVED
1 SRMVIPSE DS    CL1                     PUBLIC SPACE EXCESS IND
1 SRMVIPEN EQU   1                       - NOT PERMITTED
```

```
1 SRMVIPET EQU   2                        - TEMPORARY
1 SRMVIPEY EQU   3                        - PERMITTED
1 SRMVTPIG DS    CL1                      TPIGNORE  INDICATOR
1 SRMVTPIY EQU   1                        - YES : ERROR MESSAGES IGNORED
1 SRMVTPIN EQU   2                        - NO  : NO TPIGNORE PRIVILEGE
1 SRMVTPIR EQU   3                        - READ: ERROR MSG IGNORED-INPU
1 SRMVTPIB EQU   4                        - BLP : BY-PASS-LABEL
1 SRMVTPIA EQU   5                        - ALL : BLP AND YES PRIVILEGES
1 SRMVDFCT DS    CL4                      DEFAULT CAT-ID OF PVS
1          DS    CL4                      NOT USED
1 SRMVTSPL DS    F                        TEMP SPACE LIMIT
1 SRMVTSPU DS    F                        TEMP SPACE USED
1 SRMVFILI DS    F                        FILE LIMIT
1 SRMVFILA DS    F                        FILE AMOUNT
1 SRMVJVLI DS    F                        JV LIMIT
1 SRMVJVA  DS    F                        JV AMOUNT
1 SRMVEXHC DS    CL8                      EXTENDED HOST CODE
1 SRMVDMTR DS    X                        DMS TUNING RESOURCES
1 SRMVDTRN EQU   1                            NONE
1 SRMVDTRP EQU   2                            PAGEABLE
1 SRMVDTRR EQU   3                            RESIDENT
1 SRMVPAL  DS    X                        PHYSICAL ALLOCATION
1 SRMVPALN EQU   1                            NOT-ALLOWED
1 SRMVPALA EQU   2                            ALLOWED
1          DS    XL2                        UNUSED
1 SRMVUS1L DS    F                        S1 LEVEL USED
1 SRMVUS2L DS    F                        S2 LEVEL USED
1 SRMVDMCL DS    CL8                      DEFAULT MANAGEMENT CLASS
1 SRMVDSCL DS    CL8                      DEFAULT STORAGE CLASS
1 *                                       PERM SPACE LIMIT
1 SRMVLTOT DS    F                          TOTAL SPACE
1 SRMVLSTD DS    F                          S0 LEVEL SPACE
1 SRMVLSHP DS    F                          HIGH PERF SPACE
1 SRMVLSVP DS    F                          VERY HIGH PERF SPACE
1 SRMVLSHA DS    F                          HIGH AVAIL SPACE
1 *                                       TEMP SPACE LIMIT
1 SRMVLTMP DS    F                          TOTAL SPACE
1 SRMVLTHP DS    F                          HIGH PERF SPACE
1 SRMVLTVP DS    F                          VERY HIGH PERF SPACE
1 *                                       WORK SPACE LIMIT
1 SRMVLWRK DS    F                          TOTAL SPACE
1 SRMVLWHP DS    F                          HIGH PERF SPACE
1 SRMVLWVP DS    F                          VERY HIGH PERF SPACE
1 *                                       PERM SPACE USED
1 SRMVUTOT DS    F                          TOTAL SPACE
1 SRMVUSTD DS    F                          S0 LEVEL SPACE
1 SRMVUSHP DS    F                          HIGH PERF SPACE
1 SRMVUSVP DS    F                          VERY HIGH PERF SPACE
```

```
1 SRMVUSHA DS    F                            HIGH AVAIL SPACE
1 *                                         TEMP SPACE USED
1 SRMVUTMP DS    F                            TOTAL SPACE
1 SRMVUTHP DS    F                            HIGH PERF SPACE
1 SRMVUTVP DS    F                            VERY HIGH PERF SPACE
1 *                                         WORK SPACE USED
1 SRMVUWRK DS    F                            TOTAL SPACE
1 SRMVUWHP DS    F                            HIGH PERF SPACE
1 SRMVUWVP DS    F                            VERY HIGH PERF SPACE
1 SRMVCYSL DS    F                          CRYPTO SESSION LIMIT
1 SRMVCYSU DS    F                          CRYPTO SESSION USED
1 SRMVNST  DS    X                          NET-STORAGE USAGE
1 SRMVNSTN EQU   1                              NOT-ALLOWED
1 SRMVNSTA EQU   2                              ALLOWED
1 SRMVDMS# EQU   *-SRMVDMSD                   LENGTH OF DMS PART

1 ***************************************************************
1 *        INFORMATION ON TASK/PROGRAM                          *
1 ***************************************************************
1 SRMVTASK DS    0F
1 SRMVUSW  DS    CL4                        USER'S SWITCHES
1 SRMVAIDR DS    CL1                        AID'S READ VALUE
1 SRMVAIDW DS    CL1                        AID'S WRITE VALUE
1 SRMVTPRV DS    CL1                        TESTPRIV INDICATOR
1 SRMVTPRY EQU   1                          - TESTPRIV IS YES
1 SRMVTPRN EQU   2                          - TESTPRIV IS NO
1 SRMVADS2 DS    CL2                        ADDRSPACE VALUE
1 SRMVADIT DS    CL1                        AUDIT INDICATOR
1 SRMVADTY EQU   1                          - AUDIT ALLOWED
1 SRMVADTN EQU   2                          - AUDIT NOT ALLOWED
1 SRMVPSW  DS    CL1                        PSWORD CMD'S RIGHT
1 SRMVPSWY EQU   1                          - PSWORD YES
1 SRMVPSWM EQU   2                          - PSWORD MOD
1 SRMVPSWN EQU   3                          - PSWORD NO
1 SRMVCSMP DS    CL1                        CSTMP-MACRO INDICATOR
1 SRMVCSMY EQU   1                          - CSTMP-MACRO ALLOWED
1 SRMVCSMN EQU   2                          - CSTMP-MACRO NOT ALLOWED
1 SRMVHWAU DS    CL1                        HARDWARE-AUDIT INDICATOR
1 SRMVHWAUY EQU   1                          - HW-AUDIT ALLOWED
1 SRMVHWAUN EQU   2                          - HW-AUDIT NOT ALLOWED
1 SRMVLKAU DS    CL1                        LINKAGE-AUDIT INDICATOR
1 SRMVLKAUY EQU   1                          - LNK-AUDIT ALLOWED
1 SRMVLKAUN EQU   2                          - LNK-AUDIT NOT ALLOWED
1 SRMVPRID DS    CL54                       PROFILE ID
1 SRMVRPAG DS    H                          NB OF RESIDENT PAGES
1 SRMVDTKL DS    CL1                        DEFAULT-MESSAGE-LANGUAGE
1 SRMVMSGS DS    CL1                        DEFAULT-MESSAGE-SEARCH
1 SRMVSTSK EQU   X'01'                      -TASK
```

```
1 SRMVSALL EQU   X'02'                   -*ALL
1 SRMVADRS DS    F                       ADDRSPACE VALUE
1 SRMVRPGS DS    F                       NB OF RESIDENT PAGES
1 SRMVTSK# EQU   *-SRMVTASK                LENGTH OF TASK PART
1 ***************************************************************
1 *        INFORMATION ON SPOOL                                 *
1 ***************************************************************
1 SRMVSPOL DS    0F
1          DS    CL8                     UNUSED
1 SRMVMAIL DS    CL64                    MAILING ADDRESS
1 SRMVSPL# EQU   *-SRMVSPOL                LENGTH OF SPOOL PART
1 *-
1 SRMVUSR# EQU   *-SRMVJOB                 LENGTH OF USER PART
1 ***************************************************************
1 *        INFORMATION ON ACCOUNTING                            *
1 ***************************************************************
1 SRMVACCT DS    0F
1 SRMVNBAC DS    H                       NUMBER OF ACCOUNT-NB
1 SRMVMARC DS    H                       MAXIMUM OF ACCOUNT RECORDS
1 *-
1 SRMVACC# EQU   *-SRMVACCT                LENGTH OF ACC GLOBAL INF.
1 *-
1 *-       INFORMATION ON ACCOUNT ENTRY
1 *-
1 SRMVACCE DS    0F
1 SRMVACT  DS    CL8                     ACCOUNT NUMBER
1 SRMVCPU  DS    F                       CPU  TIME REMAINING
1 SRMVNTL  DS    CL1                     NTL INFORMATION
1 SRMVNTLY EQU   1                       - NTL ALLOWED
1 SRMVNTLN EQU   2                       - NTL NOT ALLOWED
1 SRMVEXP  DS    CL1                     EXPRESS  INFORMATION
1 SRMVEXPY EQU   1                       - EXPRESS  ALLOWED
1 SRMVEXPN EQU   2                       - EXPRESS  NOT ALLOWED
1 SRMVSCLA DS    CL1                     SPOOL-OUT CLASS
1 SRMVPRI  DS    CL1                     HIGHEST  PRIORITY PERMITTED
1 SRMVINHD DS    CL1                     INHIBIT  DEACTIVATION IND
1 SRMVINHY EQU   1                       - INHIBIT DEACTIVATION ALLOW
1 SRMVINHN EQU   2                       - INHIBIT DEACT NOT    ALLOW
1 SRMVTYPL DS    CL1                     UPPER LIMIT OF TASK TYPE
1 SRMVTYST EQU   1                       - STD
1 SRMVTYTP EQU   2                       - TP
1 SRMVTYSY EQU   3                       - SYS
```

```
1 *************************************************************/
1 * INFORMATION FOR POSIX  VERSION V11.2                      */
1 *************************************************************/
1 SRMVPOA  DS    X                      POSIX ACCOUNTNUMBER
1 SRMVPOAY EQU   1                      1: YES POSIX ACCOUNTNUMBER
1 SRMVPOAN EQU   2                      2: NO  POSIX ACCOUNTNUMBER
1 SRMVDAC  DS    X                      DEFAULT ACCOUNTNUMBER
1 SRMVDACY EQU   1                      - YES DEFAULT ACCOUNTNUMBER
1 SRMVDACN EQU   2                      - NO DEFAULT ACCOUNTNUMBER
1 SRMVACE# EQU   *-SRMVACCE               LENGTH OF ONE ACC ENTRY
1 *-
1          DS    CL(59*SRMVACE#)
1 *
1 SRMVACM# EQU   60*SRMVACE#+SRMVACC#     MAX LENGTH OF ALL ACCOUNT
1 *-                                    ENTRIES,GLOBAL INF
SRMVALL#   EQU   *-SRMVJOB                MAX LENGTH OF A ALL ENTRY
***************************************************************
*        INFORMATION ON EMAIL V17.0                           *
***************************************************************
SRMVEMA    DS    OF
SRMVEMAL   DS    H                        EMAIL-LENGTH
SRMVEMAI   DS    CL1800                   EMAIL
SRMVEMA#   EQU   *-SRMVEMA                LENGTH OF EMAIL-INFO
1 *-
1 SRMVETR# EQU   SRMVUSR#+SRMVACM#        MAX LENGTH OF A USER ENTRY
```

Example showing output of user ID and default pubset

```
SRMUINF  START
SRMUINF  RMODE ANY
SRMUINF  AMODE ANY
         GPARMOD 31
         BALR  3,0
         BCTR  3,0
         BCTR  3,0
         USING SRMUINF,3
         SRMUINF AREA@=SRMAUS,AREA#=SRMVLLLL,INFO=*ALL,MF=M ─────────  (1)
         SRMUINF MF=E,PARAM=SRMPL ───────────────────────────────────  (2)
         CLI   SRMVMR1,SRMVMOK          * ERROR IN CALL ?
         BNE   ERROR
         MVC   USERID,SRMVUSER          * CALL OK; OUTPUT USER ID
         MVC   DEFPUB,SRMVDFCT          * AND DEFAULT PUBSET
         WROUT MSGAUSG,0
         B     ENDE
ERROR    EQU   *                        * ERROR IN CALL (RC <> 0);
         UNPK  RC(5),SRMVMR1(2)         * OUTPUT RETURN CODE
         TR    RC(4),TAB
         WROUT MSGFEHL,0
* Definitions *
TAB      DS    CL240
         DC    C'0123456789ABCDEF'
*
* Parameter area SRMUINF macro
         DS    0F
SRMPL    SRMUINF MF=C
         ORG SRMVPL
         SRMUINF MF=L
         ORG
*
* Output area SRMUINF macro
         DS    0F
SRMAUS   SRMUINF MF=C,XPAND=OUTPUT,DATA=ALL
*
* Output area if call OK
MSGAUSG  DS    0H
         DC    Y(MSGAUSGL)
         DC    CL3' '
         DC    C'SRMUINF: User ID: '
USERID   DS    CL8
         DC    C'   , Default pubset: '
DEFPUB   DS    CL4
         DC    C'<'
MSGAUSGL EQU   *-MSGAUSG
*
```

```
* Output area if call contains error
MSGFEHL  DS    0H
         DC    Y(MSGFEHLL)
         DC    CL3' '
         DC    C'Error in SRMUINF: RC = '
RC       DS    CL5
MSGFEHLL EQU   *-MSGFEHL-1
SRMVLLLL DC    Y(SRMVETR#)             * LENGTH OF OUTPUT AREA
         END
               =AL4(SRMAUS)
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,srmuinf), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,srmuinf))
%  ASS6011 ASSEMBLY TIME: 438 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 151 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=srmuinf
%  BLS0523 ELEMENT 'SRMUINF', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'SRMUINF', VERSION ' ' OF '<date> <time>' LOADED
SRMUINF: User ID: QM212      , Default pubset: 2OSG<
```

(1) Initialize parameter list.

(2) Read user information.

STAMCE – Read MRSCAT entries

General

Application area:	Multiprocessor systems; see page 164
Macro type:	Type S, MF format **2**: standard/E/L/C/D/M form; see page 29

Each pubset contains a file catalog TSOSCAT which can be uniquely identified by means of its catalog ID. Furthermore, each TSOSCAT (and thus each pubset) is uniquely assigned to one system (BS2000 native or VM2000 guest system).
The TSOSCAT of each home pubset contains the pubset's entry from the MRSCAT, a catalog of all pubsets existing in the computer network. Each pubset can contain a complete MRSCAT. An entry in the MRSCAT includes

– the catalog ID of a TSOSCAT;
– the BCAM name of the system to which this TSOSCAT is assigned;
– the device type of the corresponding pubset;
– status information relating to the pubset.
– pubset-specific operands (EAM, cache, allocator,...)

Macro description

The **STAMCE** macro outputs an extract from one selected entry, or all entries, of the MRS catalog (MRSCAT) to an output area that is either defined by the user or created by the macro. The extracts provide information about to the pubsets initialized in the computer network, relating to:

– device information (catalog ID, device type code, system's BCAM name if the pubset is managed by a remote system, etc.)
– status (home pubset, local pubset, pubset not accessible, etc.)
– pubset usage (paging, data access, ...)
– attributes (EAM operand, cache configuration, etc.)
– System administration or operator only:
number, TSNs and user IDs of all jobs occupying the pubset (either as opened files or through reservation requests (SECURE locks), number and location of the CMS buffers as per default and current setting, etc.)

Individual entries are identified via the catalog ID of the pubset. Output is restricted to entries in the MRSCAT on the home pubset.

Notes

- Both the operand list and the output area have had their layout adapted to the functional enhancements of the individual BS2000 versions. For the sake of compatibility it is also possible to generate the layouts of older versions, using the **STAMCE** macro (VERSION operand).
- If MF=M, the following restriction applies:
  If the CATID or AREA operand is specified, register R15 may not be used as the base register for the parameter list or for the fields defined via these operands.

Macro format and description of operands

STAMCE
CATID='␣' / 'catid' / addr / (r)
[,AREA=addr / (r)]
[,LENGTH=length]
,REF=NO / YES / ALL
[,HOST=*LOCAL / '*LOCAL' / *ALL / '*ALL' / 'bcam-name' / addr]
,SELECT=ALL / ACCESSIBLE / DEF_XCS_CONF / EXCLUSIVE / HSMS_SUPPORTED / INACCESSIBLE / LOCAL / LOCAL_ACCESSIBLE / MASTER_CHANGE_ERROR / PAGING / QUIET / REMOTE / REMOTE_ACCESSIBLE / SCA / SHARED / SINGLE_FEATURE / SYSTEM_MANAGED / UNUSED_VOLSETS / VOLUME_SETS / XCS_CONFIGURATED
[,PUBSET=*ALL / 'catid' / addr]
[,XPAND=PL / *ALL / MCE / OCC]
[,VERSION=5 / 3]
,MF=S / E / L / C / D / M
[,PARAM=addr / (r)]
,PREFIX=D / p
[,MACID=macid]

The operands are described below in alphabetical order.

AREA=
Specifies the address of an output area in which **STAMCE** transfers the requested MRSCAT entries. If the operand is not specified, **STAMCE** requests class 6 memory for the output (in units of 4096-byte main memory pages) and supplies the DMCEAREA and DMCEARLN fields of the operand list with the address and length (in bytes) of this output area. The caller is responsible for returning the memory area requested by the macro (**RELM** macro). The caller is responsible for returning the memory area requested by the macro (**RELM** macro).

addr
Symbolic address (name) of a field which accepts the requested MRSCAT entries. The field is aligned on a word boundary.

(r)
Register containing the address value “addr”. May be specified only in conjunction with MF=M.

CATID=
Defines the catalog IDs of the pubsets whose MRSCAT entries are to be output.

'␣'
A blank enclosed in single quotes is the default setting.
The SELECT operand can be used to provide a narrower selection of MRSCAT entries to be output.

'catid'
Either an explicitly specified catalog ID or a wildcard expression for selecting catalog IDs. The specified character string should always be enclosed in apostrophes.
The following rules apply to the “catid” format, depending on whether the catalog ID is specified explicitly or as a wildcard expression:

Catalog ID specified explicitly:
– catid is either a string one to four characters long, comprising the letters A,...,Z and the digits 0,...,9 with the strings PUB and PUBx (x = any character) being excluded
– or the character # (for the home pubset).

Catalog ID specified as a wildcard expression:
catid is a string one to four characters long, which may contain the following wildcard characters in addition to letters and digits:
* Wildcard for any (also empty) string
/ Wildcard for a single character
<s1> Wildcard for the string s1 specified in parentheses (maximum two characters)
- Negation symbol for the subsequent entries (only permissible as the first character in the expression).
The SELECT operand may be used to provide a narrower selection of the MRSCAT entries to be output.

addr
Symbolic address (name) of a field which contains either an explicitly specified catalog ID or a wildcard expression for selecting catalog IDs. In either case, the caller must enter this information in the appropriate format:

Catalog ID specified explicitly:

One of the following entries is to be placed in the field:

- a string in the form
  `[:]catid[␣ / :]`
  where catid is a string one to four characters long, comprising the letters A,...,Z and the digits 0,...,9 with the strings PUB and PUBx (where x = any character) being excluded. If catid is less than four characters it must be delimited on the right (as shown above) by a blank (␣; X'40') or by a colon (:). This delimiter may be omitted if the catalog ID is four characters long.
- the character # (for the home pubset)
  Catalog ID specified as wildcard expression:
  A string in the following format is to be placed in the field:
  `[:]catid[␣ / :]`
  where catid is a wildcard expression which, including the right delimiter if present, may be up to 256 characters long and may contain the following wildcard characters in addition to letters and digits:

*	Wildcard for any (also empty) string
/	Wildcard for a single character
<s1:s2>	Wildcard for a string which lies between s1 and s2 in the lexicographical order
<s1,...>	Wildcard for one of the strings enclosed in parentheses
-	Negation symbol for the subsequent entries (only permissible as the first character of the expression).

If the wildcard expression is less than 256 characters it must be delimited on the right, as shown above, by a blank (␣; X'40') or by a colon (:). The delimiter may only be omitted in expressions with 256 characters.

The SELECT operand may be used to provide a narrower selection of the MRSCAT entries to be output.

(r)
Register containing the address value “addr”.
May be specified only in conjunction with MF=M.

Note

If MF=M and the CATID or AREA operand is specified, register R15 may not be used as a base register for the parameter list or for the fields defined via these operands.

HOST=
From amongst the systems having common access rights to a pubset, specifies the system about whose tasks the information requested with REF=ALL is to be output.
The operand is effective only if specified in conjunction with REF=ALL, otherwise it is ignored.

***LOCAL**
'*LOCAL'
The information requested with REF=ALL is output for local tasks occupying the pubset.

***ALL**
'*ALL'
If the pubset specified in the CATID operand is shareable and the local system is the master of the pubset, the information requested with REF=ALL is output for all (local and remote) tasks occupying this pubset; otherwise, information is output only for local tasks.

'bcamname'
BCAM name of the local system or (if the local system is the master of the pubset) name of a slave system.

addr
Symbolic address (name) of a field containing one of the values '*LOCAL', '*ALL' or 'bcamname'.

LENGTH=length
Specifies the length of the output area for the requested MRSCAT entries.
The implicit length of a field specified with the AREA operand is not assumed as the length of the output area even if the LENGTH operand is not specified.
With MF=S, LENGTH may only be specified in conjunction with AREA.

length
Length (in bytes) of the field with the address “addr2”.
The minimum length of the output area depends on the number of requested MRSCAT entries and the scope of the information specified in the REF operand. The following values apply:

– An individual MRSCAT entry is requested:

When REF=NO or REF=YES the output area must be at least as long as the MRSCAT entry to be output. The layout of this entry is illustrated as a DSECT following the return codes.

When REF=ALL the minimum length l_{AREA} of the output area is determined by the following formula:

$l_{AREA} = l_{MCE} + n_{OCC} * l_{OCC} + 1$

Where:

l_{AREA}	Minimum length of the output area (in bytes)
l_{MCE}	Length of the MRSCAT entry to be output (for layout see DSECT following description of return codes)
n_{OCC}	Number of occupation entries for the pubset to which the MRSCAT entry refers
l_{OCC}	Length of an occupation entry (for layout see DSECT following description of return codes)

– Several entries are requested:
The following formula applies for the minimum length l_{AREA} of the output area:

$l_{AREA} = n_{MCE} * l_{MCE} + 4$

Where:

l_{AREA}	Minimum length of the output area (in bytes)
n_{MCE}	Number of MRSCAT entries to be output
l_{MCE}	Length of the MRSCAT entry to be output (for layout see DSECT following description of return codes)

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29.
The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

Default values:	MACID = MCE if XPAND=PL
	MACID = MCF if XPAND=MCE
	MACID = MCH if XPAND=OCC

PUBSET=
May only be specified in conjunction with SELECT=VOLUME_SETS .
Specifies the catid of the SM pubset whose volume set is returned.

***ALL**
The volume sets of all SM pubsets are returned.

'catid'
Catid of the SM pubsets for whose volume sets information is is to be returned.

addr
Symbolic address (name) of a field containing the catalog id of the SM pubset for whose volume sets information is to be returned.

REF=
Determines whether further information about the specified pubsets is to be output in addition to the MRSCAT entries (in other words in addition to the information which the /SHOW-MASTER-CATALOG-ENTRY command supplies).

NO
Default setting: no further information is to be output.

YES
In addition to the MRSCAT entry, the pubset parameters are also output (see the SHOW-PUBSET-PARAMETERS command). Only the occupying tasks are missing; these are only output when REF=ALL.

This entry is interpreted for macro calls only under the system administration ID (TSOS); it is ignored for all other users.

ALL
In addition to the information described under REF=YES, the user IDs, the TSNs and the TIDs of the tasks occupying the pubset are output for a pubset, as well as the SYSIDs of the systems on which these tasks are running. The entries output are sorted for each system: firstly according to user ID, then according to TSN and finally according to TID.
REF=ALL is only permissible if the catalog ID of a pubset is specified explicitly in the CATID operand. When a wildcard expression or the (default) value '␣' is specified in the CATID operand, REF=ALL is converted internally into REF=YES.

This entry is interpreted for macro calls only under the system administration ID (TSOS); it is ignored for all other users.

SELECT=
When a wildcard expression or the (default) value '␣' (blank) is specified in the CATID operand, permits a narrower selection of the pubset thus specified.

ALL
Default setting: the requested information is output for all the pubsets specified in the CATID operand.

ACCESSIBLE
The requested information is output only for pubsets whose file catalog can be accessed.

DEF_XCS_CONF
The requested information is output only for pubsets defined as XCS pubset.

EXCLUSIVE
The requested information is output only for pubsets which are not imported as shared pubsets.

HSMS_SUPPORTED
The requested information is output only for SM pubsets to which the HSMS SUPPORTED attribute is applicable.

INACCESSIBLE
The requested information is output only for pubsets which are not imported.

LOCAL
The requested information is output only for pubsets which are imported locally.

LOCAL_ACCESSIBLE
The requested information is output only for pubsets which are imported locally and do not have quiet status.

MASTER_CHANGE_ERROR
The requested information is output only for shared pubsets in which a master change was terminated with errors.

PAGING
The requested information is output only for pubsets with paging areas which are used locally.

QUIET
The requested information is output only for pubsets which are in the quiet status.

REMOTE
The requested information is output only for pubsets where the LOCAL selection criterion is not applicable.

REMOTE_ACCESSIBLE
The requested information is output only for pubsets which are not imported locally, but whose catalog can still be accessed because an MSCF connection to a remote system which has imported the pubset locally currently exists.

SCA
The requested information is output only for pubsets for which the local system handles the catalog accesses via SCA.
SCA = Speed Catalog Access. This replaces sequential access to the TSOSCAT with indexed sequential (direct) file access.

SHARED
The requested information is output only for pubsets which are imported as shared pubsets.

SINGLE_FEATURE
The requested information is output only for SF pubsets.

SYSTEM_MANAGED
The requested information is output only for SM pubsets.

UNUSED_VOLSETS
The requested information is output only for volume sets which are in the DEFINED ONLY status.

VOLUME_SETS
The requested information is output only for volume sets.
The selection of volume set entries to be output can be narrowed with the PUBSET operand.

XCS_CONFIGURATED
The requested information is output only for pubsets which are used as an XCS pubset.

VERSION=
Specifies for which BS2000 version the layout is to be created and information output.
If the VERSION operand is not specified, layout and information for BS2000 Version V10.0 are created or output.

5
Layout and information are created or output for BS2000/OSD-BC as of V3.0.

3
Layout and information for Versions BS2000/OSD-BC V1.0 and V2.0 are created or output.

XPAND=
Only interpreted in conjunction with MF=C or MF=D and determines for which data area a CSECT or DSECT is to be generated.

PL
A CSECT/DSECT is generated for the parameter area. This is the default setting when MF=C.

***ALL**
CSECTs/DSECTs are generated for the parameter area and the output area with the occupation entry. This is the default setting when MF=D.

MCE
A CSECT/DSECT is generated for an MRSCAT entry.

OCC
A CSECT/DSECT is generated for the additional occupation information output with REF=ALL.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the STAMCE macro is transferred in the standard header:
(cc= Subcode2, bb=Subcode1, aaaa=Maincode)

Subcode2 (X'cc') is not shown in the table below. It may assume the values X'00' and X'01'. They have the following meanings:

X'00': Error in local system
X'01': Only for REF=ALL: error in remote system

X'bb'	X'aaaa'	Meaning
X'00'	X'0000'	Function executed successfully; no error.
X'20'	X'0310'	System error: error during privilege check.
X'01'	X'0311'	Operand error: – The catalog ID and an address with the catalog ID were specified – The output area is not aligned on a word boundary – Not all parts of the parameter area can be accessed – The output area is not available – The address with the catalog ID is not available, or a valid catalog ID or wildcard expression could not be found in the assigned memory area
X'40'	X'0312'	MRSCAT entry being sought not found: – There is no MRSCAT entry with the specified catalog ID – There is no MRSCAT entry corresponding to the selection criteria specified in the wildcard expression and/or in the SELECT operand
X'20'	X'0313'	Error with REQM/RELM: memory space for the output could not be requested/ released
X'40'	X'0313'	Error with REQM/RELM: insufficient class 6 memory available
X'01'	X'0314'	Catalog ID / wildcard expression invalid: The specified catalog ID or wildcard string does not correspond to the required format
X'40'	X'0316'	Output area too small: the output area specified by the user is too small to accept the requested information
X'40'	X'0317'	System error: conflict with MRSCAT lock
X'20'	X'0318'	System error: synchronization error
X'40'	X'031A'	MRSCAT not yet initialized: STAMCE was called during startup before the MRSCAT was initialized
X'20'	X'031B'	Transmission error, occurs only with REF=ALL: An error occurred during HIPLEX MSCF transmission when the pubset was requested or released

X'bb'	X'aaaa'	Meaning
X'01'	X'031C'	Invalid BCAM name, occurs only with REF=ALL: – The specified BCAM name is not known to the system – The specified BCAM name does not identify a pubset sharer – The pubset is not multiprocessor shareable and the BCAM name does not identify the local system. – The local system is not the master of the multiprocessor shareable pubset and the specified BCAM name does not identify the local system
X'01'	X'031F'	Incorrect SELECT operand in the operand list: The user has supplied the field for the SELECT operand in the operand list with an invalid value
X'01'	X'FFFF'	Interface error: incorrect unit or function code
X'03'	X'FFFF'	Interface error: incorrect version number

The following values may be returned as additional return codes in the rightmost two bytes of register 15:

X'04A4': An attempt was made to execute a **STAMCE** of a version < V9.5 and an invalid operation code was detected in the operand list.
This error will occur particularly when the version of the operand list is incompatible with the version of the SVC call.

X'04A0': As above; in addition, the JV (“Job Variables”) subsystem is not available.

Other return codes which, in accordance with conventions, apply to all macros are given in table “Standard return codes” on page 43.

The calling program is terminated when the following errors occur:

- The data area is not assigned to the caller.
- The data area is not aligned on a word boundary.
- The data area is protected against write access.

DSECT for the operand list of the macro (XPAND=PL)

```
          STAMCE MF=D,PREFIX=D,XPAND=PL,VERSION=5
1         #INTF REFTYPE=REQUEST,                                             C
1               INTNAME=STAM,                                                C
1               INTCOMP=5
1         MFCHK    MF=D,                                                     C
1               SUPPORT=(C,D,E,L,M,S),                                       C
1               PREFIX=D,                                                    C
1               MACID=MCE,                                                   C
1               DMACID=MCE,                                                  C
1               DNAME=MCEPL,                                                 C
1               PARAM=,                                                      C
1               SVC=33,                                                      C
1               ALIGN=F
2 DMCEPL   DSECT ,
2               *,##### PREFIX=D, MACID=MCE #####
1         STAMLY   MF=D,                                                     C
1               PREFIX=D,                                                    C
1               MACID=MCE,                                                   C
1               PARAM=,                                                      C
1               VERSION=5,                                                   C
1               XPAND=PL,                                                    C
1               FUNCT=1,                                                     C
1               CG27=DMCE,                                                   C
1               CATID=,                                                      C
1               AREA=,                                                       C
1               LENGTH=,                                                     C
1               REF=,                                                        C
1               HOST=,                                                       C
1               SELECT=,                                                     C
1               PUBSET=
2         #INTF REFTYPE=REQUEST,                                             C
2               INTNAME=STAMLY,                                              C
2               INTCOMP=5
2 *******************
2 *  parameter list  *
2 *******************
2 DMCEFHDR   FHDR  MF=(C,DMCE),EQUATES=NO                standard header
3 DMCEFHDR DS    0A
3 DMCEFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
3 *
3 DMCEIFID DS    0A              0   INTERFACE IDENTIFIER
3 DMCEFCTU DS    AL2             0   FUNCTION UNIT NUMBER
3 *                                  BIT 15    HEADER FLAG BIT,
3 *                                  MUST BE RESET UNTIL FURTHER NOTICE
3 *                                  BIT 14-12 UNUSED, MUST BE RESET
3 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
```

```
3 DMCEFCT  DS    AL1             2   FUNCTION NUMBER
3 DMCEFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
3 *
3 DMCERET  DS    0A              4   GENERAL RETURN CODE
3 DMCESRET DS    0AL2            4   SUB RETURN CODE
3 DMCESR2  DS    AL1             4   SUB RETURN CODE 2
3 DMCESR1  DS    AL1             5   SUB RETURN CODE 1
3 DMCEMRET DS    0AL2            6   MAIN RETURN CODE
3 DMCEMR2  DS    AL1             6   MAIN RETURN CODE 2
3 DMCEMR1  DS    AL1             7   MAIN RETURN CODE 1
3 DMCEFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
3 *
2 DMCEAREA   DS    A                area address
2 DMCECTAD   DS    A                catid address
2 DMCECTID   DS    CL4              catid
2 DMCEARLN   DS    F                area length
2 DMCERESA   DS    A                reserved (mf=m buffer)
2 DMCEHOST   DS    CL8              bcam name of the host
2 DMCESLCT   DS    C                SELECT value
2 DMCEFLAG   DS    X                flags
2 DMCESMPU   DS    CL4              id of sm pubset
2 DMCEUNUS   DS    XL6              unused
2 DMCE#      EQU   *-DMCEFHDR         length parameter list
2 *
2 * return codes
2 *
2 DMCEOK     EQU   X'0000'          NO_ERROR
2 DMCESRPM   EQU   X'0310'          SRPM_ERROR
2 DMCEOPER   EQU   X'0311'          OPERAND_ERROR_IN_STAM
2 DMCENFND   EQU   X'0312'          MCE_CANNOT_BE_FOUND
2 DMCERQRL   EQU   X'0313'          REQM_RELM_ERROR
2 DMCEICOW   EQU   X'0314'          INVALID_CATID_OR_WILDCARD
2 DMCEATS    EQU   X'0316'          AREA_TOO_SMALL
2 DMCELCKC   EQU   X'0317'          MRSCAT_LOCK_CONFLICT
2 DMCESYER   EQU   X'0318'          SYNCHRONIZATION_ERROR
2 DMCENINI   EQU   X'031A'          MRSCAT_NOT_INITIALIZED
2 DMCETRER   EQU   X'031B'          TRANSMISSION_ERROR
2 DMCEHOIN   EQU   X'031C'          INVALID_HOST_NAME
2 DMCEPAER   EQU   X'031F'          MRS_PARAM_ERROR
2 *
2 * subcode 2
2 *
2 DMCESC2L   EQU  X'00'             LOCAL_ERROR
2 DMCESC2R   EQU  X'01'             REMOTE_ERROR
2 *
2 * last catid
2 *
2 DMCELAST   EQU  X'40404040'
```

```
2 *
2 * last occupation
2 *
2 DMCELOCC   EQU  X'00'
2 *
2 * SELECT values
2 *
2 DMCEALL    EQU   0                all the pubsets
2 DMCEPAGI   EQU   1                paging pubsets
2 DMCELOCA   EQU   2                local pubsets
2 DMCEREMO   EQU   3                remote pubsets
2 DMCEACCE   EQU   4                pubsets accessible
2 DMCELOAC   EQU   5                pubsets local and accessible
2 DMCESHAR   EQU   6                shared pubsets
2 DMCEEXCL   EQU   7                exclusive pubsets
2 DMCEREAC   EQU   8                pubsets remote and accessible
2 DMCESCA    EQU   9                local and accessible pubsets which
2 *                                 are connected to SPEEDCAT
2 DMCEXCS    EQU   10               XCS configurated pubsets
2 DMCEHSMS   EQU   11               all HSMS supported sm pubsets
2 DMCESF     EQU   12               all single feature pubsets
2 DMCESM     EQU   13               all system managed pubsets
2 DMCEVOL    EQU   14               all volume sets of a sm pubset
2 DMCEDEF    EQU   15               volume sets of sm pubsets which are
2 *                                 defined but not in use
2 DMCEMCHE   EQU   16               pubsets where an error occured during
2 *                                 master change processing
2 DMCEINAC   EQU   17               pubsets which are inaccessible
2 DMCEDXCS   EQU   18               defined XCS configurated
2 DMCEQUIT   EQU   19               pubset in quiet status
2 *
2 * flags
2 *
2 DMCEMPVS   EQU  X'80'             set: MPVS mode
2 DMCEREF    EQU  X'40'             set: REF = YES
2 DMCERALL   EQU  X'20'             set: REF = ALL
```

DSECT for the output area of the macro (XPAND=MCE)

```
           STAMCE MF=D,PREFIX=D,XPAND=MCE,VERSION=5
1          #INTF REFTYPE=REQUEST,                                        C
1                INTNAME=STAM,                                           C
1                INTCOMP=5
1          MFCHK   MF=D,                                                 C
1                SUPPORT=(C,D,E,L,M,S),                                  C
1                PREFIX=D,                                               C
1                MACID=MCF,                                              C
1                DMACID=MCF,                                             C
1                DNAME=MCFMCE,                                           C
1                PARAM=,                                                 C
1                SVC=33,                                                 C
1                ALIGN=F
2 DMCFMCE  DSECT ,
2                *,##### PREFIX=D, MACID=MCF #####
1          STAMLY   MF=D,                                                C
1                PREFIX=D,                                               C
1                MACID=MCF,                                              C
1                PARAM=,                                                 C
1                VERSION=5,                                              C
1                XPAND=MCE,                                              C
1                FUNCT=1,                                                C
1                CG27=DMCF,                                              C
1                CATID=,                                                 C
1                AREA=,                                                  C
1                LENGTH=,                                                C
1                REF=,                                                   C
1                HOST=,                                                  C
1                SELECT=,                                                C
1                PUBSET=
2          #INTF REFTYPE=REQUEST,                                        C
2                INTNAME=STAMLY,                                         C
2                INTCOMP=5
2 DMCFMST    DS    0D
2          MCEDSK  VERSION=5,CG27=DMCF
3          #INTF REFTYPE=REQUEST,                                        C
3                INTNAME=MCEDSK,                                         C
3                INTCOMP=5
3 *******************************
3 *  header of all entry types  *
3 *******************************
3            DS    0D       DW alignement
3 DMCFSHDR   DS    0XL8     header
3 DMCFSCTD   DS    CL4      catid
3 *
3 DMCFSENT   DS    X        entry type set
```

```
3 DMCFSSF    EQU   0        single feature pubset
3 DMCFSSM    EQU   1        system managed pubset
3 DMCFSVOL   EQU   2        volume set
3 *
3            DS    XL3      unused
3 ********************************************
3 *  starting point of the static entry part  *
3 ********************************************
3 DMCFSDSC   DS    0D       starting point of static part
3 *************************************************
3 *  static part of a single feature pubset entry  *
3 *************************************************
3 DMCFSBCA   DS    CL8      BCAM name for RFA
3 DMCFSDEM   DS    0XL2     device mnemonic
3 DMCFSDEV   DS    XL1      device code
3 DMCFSDEF   DS    XL1      device filler
3 DMCFSBNU   DS    XL2      static number of CMS buffer
3 DMCFSBWT   DS    F        batch wait time
3 DMCFSDWT   DS    F        dialog wait time
3 *
3 * static pubset status values (1)
3 *
3 DMCFSSTA   DS    X        static status byte (1)
3 DMCFSAUT   EQU   X'80'    set  : autoquiet selected
3 DMCFSBDF   EQU   X'40'    set  : static buffer defined
3 DMCFSBCL   EQU   X'20'    set  : static buffer resident (cl.3)
3 *                         reset: static buffer non-res. (cl.4)
3 DMCFSSH    EQU   X'10'    set  : pubset shared (next import)
3 *                         reset: pubset exclusive (next import)
3 DMCFSAC    EQU   X'08'    set  : pubset with controlled use
3 DMCFSUVA   EQU   X'04'    set  : physical allocation by users allowed
3 DMCFSFIM   EQU   X'02'    set  : continue IMPORT if cache can't be
3 *                                connected
3 *                         reset: abort IMPORT if cache can't be
3 *                                connected
3 DMCFSXCS   EQU   X'01'    set  : XCS pubset at startup
3 *
3 DMCFSSPC   DS    X        SPEEDCAT set
3 DMCFSNSP   EQU   0        no automatic start
3 DMCFSSSP   EQU   1        SPEEDCAT task
3 DMCFSUSP   EQU   2        SPEEDCAT own task
3 DMCFSNSS   EQU   4        no, non start SPEEDCAT
3 *
3 * static pubset status values (2)
3 *
3 DMCFSST2   DS    X        static status byte (2)
3 DMCFSRIM   EQU   X'80'    set  : remote import by command only
3 *                         reset: remote import by connection
```

```
3 DMCFSCCT   EQU   X'40'    set  : TSOSCAT is to be converted to V10
3 *                                format during next EXPORT
3            DS    XL1      unused
3 DMCFSUID   DS    CL8      userid allowed to access pubset
3 *
3 * static cache values
3 *
3 DMCFSCBS   DS    F        cache size
3 *
3 DMCFSCBY   DS    X        byte for bit values
3 DMCFSCBU   EQU   X'80'    set  : cache size unit is KB
3 *                         reset: cache size unit is MB
3 DMCFSCST   EQU   X'40'    set  : size tolerance
3 *
3 DMCFSCM    DS    X        cache medium set
3 DMCFSCNC   EQU   0        no cache
3 DMCFSCDC   EQU   1        controller
3 DMCFSCES   EQU   2        expanded storage
3 DMCFSCGS   EQU   3        global storage
3 DMCFSCMM   EQU   4        main memory
3 *
3 DMCFSCSZ   DS    X        segment size set
3 DMCFSC4    EQU   0        4 KB
3 DMCFSC8    EQU   1        8 KB
3 DMCFSC16   EQU   2        16 KB
3 DMCFSC32   EQU   3        32 KB
3 *
3 DMCFSCFS   DS    X        file selection
3 DMCFSBUS   EQU   0        by user
3 DMCFSALL   EQU   1        all
3 DMCFSAUS   EQU   2        autoselected
3 *
3 DMCFSGDS   DS    X        GS data security set
3 DMCFSGNS   EQU   0        no security
3 DMCFSGCO   EQU   1        connect
3 *
3 DMCFSGU1   DS    XL1      GS unit=1 or 2
3 DMCFSGU2   DS    XL1      unused
3 *
3 DMCFSGDB   DS    X        GS double recording by buffer set
3 DMCFSGST   EQU   0        std
3 DMCFSGNY   EQU   1        mono
3 DMCFSGNN   EQU   2        any
3 DMCFSGYE   EQU   3        yes
3 *
3 DMCFSGFO   DS    X        force out set
3 DMCFSGNF   EQU   0        no force out
3 DMCFSGLF   EQU   1        at low filling
```

```
3 DMCFSGHF   EQU   2        at high filling
3 *
3 DMCFSCFE   DS    X        DC prefetch set
3 DMCFSCFN   EQU   0        no prefetch
3 DMCFSCFL   EQU   1        low
3 DMCFSCFH   EQU   2        high
3 *
3 * static attach/detach pubset
3 *
3 DMCFSMN    DS    XL2      attach pubset with dev#
3 *
3 * static allocator values
3 *
3 DMCFSAL1   DS    F        residual space at sat level 1
3 DMCFSAL2   DS    F        residual space at sat level 2
3 DMCFSAL3   DS    F        residual space at sat level 3
3 DMCFSAL4   DS    F        residual space at sat level 4
3 DMCFSAL5   DS    F        residual space at sat level 5
3 DMCFSAPA   DS    F        predet primary alloc amount
3 DMCFSASA   DS    F        predet secondary alloc amount
3 DMCFSADL   DS    F        sec alloc doubling limit
3 DMCFSAZP   DS    F        residual space for ZIP startup
3 *
3 * static eam values
3 *
3 DMCFSEMA   DS    F        maximal size of file SYSEAM
3 DMCFSEMI   DS    F        minimal size of file SYSEAM
3 DMCFSESA   DS    F        secondary allocation of file SYSEAM
3 DMCFSEMS   DS    F        virtual memory size of file SYSEAM
3 *
3 DMCFS#     EQU   *-DMCFSBCA    length of the static part of a single
3 *                              feature pubset entry
3            ORG   DMCFSDSC      starting point of static part
3 **************************************************
3 *  static part of a system managed pubset entry  *
3 **************************************************
3 DMCFHBCA   DS    CL8      BCAM name for RFA
3 DMCFHDEM   DS    0XL2     device mnemonic of volres of ctl volset
3 DMCFHDEV   DS    XL1      device code
3 DMCFHDEF   DS    XL1      device filler
3 DMCFHBNU   DS    XL2      static number of CMS buffer
3 DMCFHBWT   DS    F        batch wait time
3 DMCFHDWT   DS    F        dialog wait time
3 *
3 * static pubset status values (1)
3 *
3 DMCFHSTA   DS    X        static status byte (1)
3 DMCFHAUT   EQU   X'80'    set  : autoquiet selected
```

```
3 DMCFHBDF   EQU   X'40'    set  : static buffer defined
3 DMCFHBCL   EQU   X'20'    set  : static buffer resident (cl.3)
3 *                         reset: static buffer non-res. (cl.4)
3 DMCFHSH    EQU   X'10'    set  : pubset shared (next import)
3 *                         reset: pubset exclusive (next import)
3 DMCFHAC    EQU   X'08'    set  : pubset with controlled use
3 DMCFHHSM   EQU   X'04'    set  : pubset is HSMS supported
3 DMCFHFIM   EQU   X'02'    set  : continue IMPORT if cache can't be
3 *                                connected
3 *                         reset: abort IMPORT if cache can't be
3 *                                connected
3 DMCFHXCS   EQU   X'01'    set  : XCS pubset at startup
3 *
3            DS    XL1      unused
3 *
3 * static pubset status values (2)
3 *
3 DMCFHST2   DS    X        static status byte (2)
3 DMCFHRIM   EQU   X'80'    set  : remote import by command only
3 *                         reset: remote import by connection
3            DS    XL1      unused
3 DMCFHUID   DS    CL8      userid allowed to access pubset
3 *
3 * static cache values
3 *
3            DS    XL4      unused
3 DMCFHCBY   DS    X        byte for bit values
3 DMCFHCST   EQU   X'40'    set  : size tolerance
3 *
3 DMCFHVID   DS    CL4      catid of ctl volset
3 *
3 DMCFHDFF   DS    X        default file format set
3 DMCFHDST   EQU   0        std
3 DMCFHPAM   EQU   1        pamkey format
3 DMCFHNK2   EQU   2        NK2 format
3 DMCFHNK4   EQU   3        NK4 format
3 *
3            DS    XL4      unused
3 *
3 * static attach/detach pubset
3 *
3 DMCFHMN    DS    XL2      attach pubset with dev#
3 *
3 * static allocator values
3 *
3            DS    XL20     unused
3 DMCFHAPA   DS    F        predet primary alloc amount
3 DMCFHASA   DS    F        predet secondary alloc amount
```

```
3 DMCFHADL   DS    F        sec alloc doubling limit
3            DS    XL4      unused
3 *
3 * static EAM values
3 *
3 DMCFHEMA   DS    F        maximal size of file SYSEAM
3 DMCFHEMI   DS    F        minimal size of file SYSEAM
3 DMCFHESA   DS    F        secondary allocation of file SYSEAM
3 DMCFHEMS   DS    F        virtual memory size of file SYSEAM
3 *
3 DMCFH#     EQU   *-DMCFHBCA    length of the static part of a system
3 *                              managed pubset entry
3            ORG   DMCFSDSC      starting point of static part
3 ***********************************
3 *  static part of a pubset entry  *
3 ***********************************
3 DMCFFBCA   DS    CL8      BCAM name for RFA
3 DMCFFDEM   DS    0XL2     device mnemonic
3 DMCFFDEV   DS    XL1      device code
3 DMCFFDEF   DS    XL1      device filler
3 DMCFFBNU   DS    XL2      static number of CMS buffer
3 DMCFFBWT   DS    F        batch wait time
3 DMCFFDWT   DS    F        dialog wait time
3 *
3 * static pubset status values (1)
3 *
3 DMCFFSTA   DS    X        static status byte (1)
3 DMCFFAUT   EQU   X'80'    set  : autoquiet selected
3 DMCFFBDF   EQU   X'40'    set  : static buffer defined
3 DMCFFBCL   EQU   X'20'    set  : static buffer resident (cl.3)
3 *                         reset: static buffer non-res. (cl.4)
3 DMCFFSH    EQU   X'10'    set  : pubset shared (next import)
3 *                         reset: pubset exclusive (next import)
3 DMCFFAC    EQU   X'08'    set  : pubset with controlled use
3 DMCFFUVA   EQU   X'04'    set  : physical allocation by users allowed
3 DMCFFFIM   EQU   X'02'    set  : continue IMPORT if cache can't be
3 *                                connected
3 *                         reset: abort IMPORT if cache can't be
3 *                                connected
3 DMCFFXCS   EQU   X'01'    set  : XCS pubset at startup
3            DS    XL1      unused
3 *
3 * static pubset status values (2)
3 *
3 DMCFFST2   DS    X        static status byte (2)
3 DMCFFRIM   EQU   X'80'    set  : remote import by command only
3 *                         reset: remote import by connection
3            DS    XL1      unused
```

```
3 DMCFFUID   DS    CL8      userid allowed to access pubset
3 *
3 * static cache values
3 *
3            DS    XL4      unused
3 DMCFFCBY   DS    X        byte for bit values
3 DMCFFCST   EQU   X'40'    set  : size tolerance
3 *
3            DS    XL9      unused
3 *
3 * static attach/detach pubset
3 *
3 DMCFFMN    DS    XL2      attach pubset with dev#
3 *
3 * static allocator values
3 *
3            DS    XL20     unused
3 DMCFFAPA   DS    F        predet primary alloc amount
3 DMCFFASA   DS    F        predet secondary alloc amount
3 DMCFFADL   DS    F        sec alloc doubling limit
3            DS    XL4      unused
3 *
3 * static eam values
3 *
3 DMCFFEMA   DS    F        maximal size of file SYSEAM
3 DMCFFEMI   DS    F        minimal size of file SYSEAM
3 DMCFFESA   DS    F        secondary allocation of file SYSEAM
3 DMCFFEMS   DS    F        virtual memory size of file SYSEAM
3 *
3 DMCFF#     EQU   *-DMCFFBCA    length of the static part of a pubset
3 *                              entry
3            ORG   DMCFSDSC      starting point of static part
3 ***************************************
3 *  static part of a volume set entry  *
3 ***************************************
3 DMCFBPID   DS    CL4      corresponding pubset id
3            DS    XL4      unused
3 DMCFBDEM   DS    0XL2     device mnemonic
3 DMCFBDEV   DS    XL1      device code
3 DMCFBDEF   DS    XL1      device filler
3 *
3 * static performance attributes
3 *
3 DMCFBVSU   DS    X        volset usage set
3 DMCFBVST   EQU   0        standard volset
3 DMCFBWRK   EQU   1        work volset
3 DMCFBHSM   EQU   2        HSMS controlled volset
3 *
```

```
3 DMCFBAVA   DS    X        availability set
3 DMCFBAST   EQU   0        standard availability
3 DMCFBHIG   EQU   1        high availability
3 *
3 DMCFBPER   DS    X        performance profile
3 DMCFBPST   EQU   X'80'    standard performance
3 DMCFBHIH   EQU   X'40'    high performance
3 DMCFBVHI   EQU   X'20'    very high performance
3 *
3 DMCFBCRE   DS    X        write consistency set
3 DMCFBBYC   EQU   0        by close
3 DMCFBIMM   EQU   1        immediate
3 *
3            DS    XL1      unused
3 *
3 DMCFBNFA   DS    X        new file allocation
3 DMCFBNNR   EQU   0        not restricted
3 DMCFBNPO   EQU   1        physical only
3 DMCFBNNA   EQU   2        not allowed
3 *
3 DMCFBVAC   DS    X        volset access
3 DMCFBVNR   EQU   0        not restricted
3 DMCFBVAO   EQU   1        administrator only
3 *
3 DMCFBVSS   DS    X        volset status
3 DMCFBVSD   EQU   0        normal use
3 DMCFBVDO   EQU   1        defined only
3 DMCFBVIH   EQU   2        in hold
3 DMCFBVDF   EQU   3        defect
3 *
3            DS    XL2      unused
3 *
3 * static volset status values
3 *
3 DMCFBSTA   DS    X        static status byte
3 DMCFBCVS   EQU   X'04'    set: volset is ctl volset of a sm pubset
3            DS    XL11     unused
3 *
3 * static cache values
3 *
3 DMCFBCBS   DS    F        cache size
3 *
3 DMCFBCBY   DS    X        byte for bit values
3 DMCFBCBU   EQU   X'80'    set  : cache size unit is KB
3 *                         reset: cache size unit is MB
3 *
3 DMCFBCM    DS    X        cache medium set
3 DMCFBCNC   EQU   0        no cache
```

```
3 DMCFBCDC   EQU   1        controller
3 DMCFBCES   EQU   2        expanded storage
3 DMCFBCGS   EQU   3        global storage
3 DMCFBCMM   EQU   4        main memory
3 *
3 DMCFBCSZ   DS    X        segment size set
3 DMCFBC4    EQU   0        4 KB
3 DMCFBC8    EQU   1        8 KB
3 DMCFBC16   EQU   2        16 KB
3 DMCFBC32   EQU   3        32 KB
3 *
3 DMCFBCFS   DS    X        file selection
3 DMCFBBUS   EQU   0        by user
3 DMCFBALL   EQU   1        all
3 DMCFBAUS   EQU   2        autoselected
3 *
3 DMCFBGDS   DS    X        GS data security set
3 DMCFBGNS   EQU   0        no security
3 DMCFBGCO   EQU   1        connect
3 *
3 DMCFBGU1   DS    XL1      GS unit=1 or 2
3 DMCFBGU2   DS    XL1      unused
3 *
3 DMCFBGDB   DS    X        GS double recording by buffer set
3 DMCFBGST   EQU   0        standard
3 DMCFBGNY   EQU   1        mono
3 DMCFBGNN   EQU   2        any
3 DMCFBGYE   EQU   3        dual
3 *
3 DMCFBGFO   DS    X        force out set
3 DMCFBGNF   EQU   0        no force out
3 DMCFBGLF   EQU   1        at low filling
3 DMCFBGHF   EQU   2        at high filling
3 *
3 DMCFBCFE   DS    X        DC prefetch set
3 DMCFBCFN   EQU   0        no prefetch
3 DMCFBCFL   EQU   1        low
3 DMCFBCFH   EQU   2        high
3 *
3 * static allocator values
3 *
3 DMCFBAL1   DS    F        residual space at sat level 1
3 DMCFBAL2   DS    F        residual space at sat level 2
3 DMCFBAL3   DS    F        residual space at sat level 3
3 DMCFBAL4   DS    F        residual space at sat level 4
3 DMCFBAL5   DS    F        residual space at sat level 5
3            DS    XL12     unused
3 DMCFBAZP   DS    F        residual space for ZIP startup
```

```
3 *
3            DS    XL16      unused
3 *
3 DMCFB#     EQU   *-DMCFBPID     length of the static part of a volume
3 *                               set entry
3            ORG   DMCFSDSC+DMCFS#      length of a static MRSCAT entry
2 **********************************************
2 *  starting point of the dynamic entry part  *
2 **********************************************
2 DMCFD      DS    0D        starting point of dynamic part
2 ***************************************************
2 *  dynamic part of a single feature pubset entry  *
2 ***************************************************
2 DMCFDOC#   DS    F         counter for occupations
2 *
2 * dynamic pubset status values (1)
2 *
2 DMCFDSTA   DS    X         dynamic status byte (1)
2 DMCFDLOC   EQU   X'80'     set: local          reset: remote
2 DMCFDHOM   EQU   X'40'     set: home           reset: imported
2 DMCFDSH    EQU   X'20'     set: shared         reset: exclusive
2 DMCFDIMC   EQU   X'10'     set: import in process
2 DMCFDEXC   EQU   X'08'     set: export in process
2 DMCFDMAS   EQU   X'04'     set: master         reset: slave
2 DMCFDINA   EQU   X'02'     set: inaccessible   reset: not inacc
2 DMCFDQUI   EQU   X'01'     set: quiet
2 *
2 * dynamic pubset status values (2)
2 *
2 DMCFDST2   DS    X         dynamic status byte (2)
2 DMCFDUVA   EQU   X'10'     set: physical allocation by users allowed
2 DMCFDAC    EQU   X'08'     set: pubset with controlled use
2 DMCFDMCP   EQU   X'04'     set: master change in process
2 DMCFDPAG   EQU   X'02'     set: paging pubset
2 DMCFDERI   EQU   X'01'     set: eram inhibit
2 *
2 DMCFDSES   DS    X         pubset session number
2 *
2 DMCFDFLA   DS    X         CMS flags
2 DMCFDBDF   EQU   X'80'     set:   CMS buffers defined
2 DMCFDBCL   EQU   X'40'     set:   CMS buffers resident (class 3)
2 *                          reset: CMS buffers not resident (class 4)
2 DMCFDSPC   EQU   X'20'     set: speedcat is running
2 DMCFDELC   EQU   X'10'     set: Extra_large_catalog
2 *
2 DMCFDBNU   DS    XL2       number of CMS buffers
2 *
2 DMCFDATT   DS    X         attribute
```

```
2 DMCFDLOB   EQU   X'40'    set: large_objects
2 *                                 files/volumes with more than 32 GB
2 DMCFDLFA   EQU   X'20'    set: large_files_allowed
2 DMCFDRAI   EQU   X'10'    set: pubset with RAID volumes
2 DMCFDGSV   EQU   X'08'    set: gs volumes
2 DMCFDDRV   EQU   X'02'    set: high availability by DRV
2 DMCFDKEY   EQU   X'01'    set: key pubset
2 *
2 DMCFDXCN   DS    CL8      XCS name
2 DMCFDHOS   DS    CL8      host name       :* MSCF host name
2 *
2 DMCFDPUB   DS    X        pubset set
2 DMCFD2KN   EQU   0        NK2 (2K native)
2 DMCFD4KN   EQU   1        NK4 (4K native)
2 DMCFD4KO   EQU   2        NK2, allocation unit multiple of 4K
2 *                         (4K oriented)
2 * dynamic cache values
2 *
2 DMCFDCSZ   DS    F        size of cache buffer
2 *
2 DMCFDCB8   DS    X        byte for bit values
2 DMCFDCBU   EQU   X'80'    set  : cache size unit is KB
2 *                         reset: cache size unit is MB
2 DMCFDCDS   EQU   X'40'    set  : data security ensured
2 DMCFDCDB   EQU   X'20'    set  : double recording by buffer
2 DMCFDCDD   EQU   X'10'    set  : cache deactivated
2 DMCFDCIH   EQU   X'08'    set  : cache in hold
2 DMCFDCCU   EQU   X'04'    set  : cache used
2 DMCFDCSF   EQU   X'02'    set  : save file failed
2 *
2 DMCFDCM    DS    X        cache medium set
2 DMCFDCNC   EQU   0        no cache
2 DMCFDCDC   EQU   1        controller
2 DMCFDCES   EQU   2        expanded storage
2 DMCFDCGS   EQU   3        global storage
2 DMCFDCMM   EQU   4        main memory
2 *
2 DMCFDCS    DS    X        segment size set
2 DMCFDC4    EQU   0        4 KB
2 DMCFDC8    EQU   1        8 KB
2 DMCFDC16   EQU   2        16 KB
2 DMCFDC32   EQU   3        32 KB
2 *
2 DMCFDCU1   DS    XL1      GS unit=1 or 2
2 DMCFDCU2   DS    XL1      unused
2 *
2 DMCFDCFO   DS    X        force out set
2 DMCFDCNF   EQU   0        no force out
```

```
2 DMCFDCIP   EQU   1        at low filling
2 DMCFDCIN   EQU   2        at high filling
2 *
2 DMCFDCFE   DS    X        prefetch set
2 DMCFDCFN   EQU   0        no prefetch
2 DMCFDCFL   EQU   1        low
2 DMCFDCFH   EQU   2        high
2 *
2 DMCFDCFS   DS    X        file selection
2 DMCFDBUS   EQU   0        by user
2 DMCFDALL   EQU   1        all
2 DMCFDAUS   EQU   2        auto select
2 *
2 DMCFDCAS   DS    H        size of allocation unit (# of half pages)
2 DMCFDMTL   DS    H        maximal I/O transfer length
2 DMCFDUID   DS    CL8      userid allowed to access pubset
2 *
2            DS    XL2      unused
2 *
2 * dynamic allocator values
2 *
2 DMCFDAL5   DS    F        residual space at sat level 5
2 DMCFDAL4   DS    F        residual space at sat level 4
2 DMCFDAL3   DS    F        residual space at sat level 3
2 DMCFDAL2   DS    F        residual space at sat level 2
2 DMCFDAL1   DS    F        residual space at sat level 1
2 DMCFDAPA   DS    F        predet primary alloc amount
2 DMCFDASA   DS    F        predet secondary alloc amount
2 DMCFDADL   DS    F        sec alloc doubling limit
2 DMCFDAZP   DS    F        residual space for ZIP startup
2 *
2 * dynamic EAM values
2 *
2 DMCFDEMA   DS    F        minimal size of file SYSEAM
2 DMCFDEMI   DS    F        maximal size of file SYSEAM
2 DMCFDESA   DS    F        secondary allocation of file SYSEAM
2 DMCFDEMS   DS    F        virtual memory size of file SYSEAM
2 *
2 DMCFDREF   DS    XL4      counter of occupations (duplicate)
2 *
2 DMCF#      EQU   *-DMCFMST      length of the STAM single
2 *                               feature pubset entry
2            ORG   DMCFD    starting point of dynamic part
2 ***************************************************
2 *  dynamic part of a system managed pubset entry  *
2 ***************************************************
2 DMCFKOC#   DS    F        counter for occupations
2 *
```

```
2 * dynamic pubset status values (1)
2 *
2 DMCFKSTA   DS    X        dynamic status byte (1)
2 DMCFKLOC   EQU   X'80'    set: local          reset: remote
2 DMCFKHOM   EQU   X'40'    set: home           reset: imported
2 DMCFKSH    EQU   X'20'    set: shared         reset: exclusive
2 DMCFKIMC   EQU   X'10'    set: import in process
2 DMCFKEXC   EQU   X'08'    set: export in process
2 DMCFKMAS   EQU   X'04'    set: master         reset: slave
2 DMCFKINA   EQU   X'02'    set: inaccessible   reset: not inacc
2 DMCFKQUI   EQU   X'01'    set: quiet
2 *
2 * dynamic pubset status values (2)
2 *
2 DMCFKST2   DS    X        dynamic status byte (2)
2 DMCFKAC    EQU   X'08'    set: pubset with controlled use
2 DMCFKMCP   EQU   X'04'    set: master change in process
2 DMCFKPAG   EQU   X'02'    set: paging pubset
2 DMCFKERI   EQU   X'01'    set: eram inhibit
2 *
2 DMCFKSES   DS    X        session number
2 *
2 DMCFKFLA   DS    X        CMS flags
2 DMCFKBDF   EQU   X'80'    set:   CMS buffers defined
2 DMCFKBCL   EQU   X'40'    set:   CMS buffers resident (class 3)
2 *                         reset: CMS buffers not resident (class 4)
2 DMCFKBNU   DS    XL2      number of CMS buffers
2 *
2 * dynamic sm pubset status values
2 *
2 DMCFKSMS   DS    X        special status bytes for sm pubsets
2 DMCFKGEN   EQU   X'80'    set: pubset is in generation
2            ORG   DMCFKSMS
2 DMCFKATT   DS    X        attribute
2 DMCFKLOB   EQU   X'40'    set: large_objects
2 *                                 files/volumes with more than 32 GB
2 DMCFKLFA   EQU   X'20'    set: large_files_allowed
2 DMCFKXCN   DS    CL8      XCS name
2 DMCFKHOS   DS    CL8      host name       :* MSCF host name
2 *
2 DMCFKDFF   DS    X        default file format set
2 DMCFKPAM   EQU   0        pamkey format
2 DMCFKNO2   EQU   1        NK2 format
2 DMCFKNO4   EQU   2        NK4 format
2 *
2 * dynamic performance attributes
2 *
2 DMCFKPER   DS    X        performance profile
```

```
2 DMCFKSTD   EQU   X'80'    standard performance
2 DMCFKHIG   EQU   X'40'    high performance
2 DMCFKVHI   EQU   X'20'    very high performance
2 *
2 DMCFKWRC   DS    X        write consistency
2 DMCFKBC    EQU   X'80'    by close
2 DMCFKIMM   EQU   X'40'    immediate
2 *
2 DMCFKAVA   DS    X        availability
2 DMCFKAST   EQU   X'80'    standard availability
2 DMCFKAHI   EQU   X'40'    high availability
2 *
2            DS    XL1      unused
2 *
2 DMCFKFMT   DS    X        format profile
2 DMCFKK     EQU   X'80'    system managed pubset with K volsets
2 DMCFKNK2   EQU   X'40'    system managed pubset with NK2 volsets
2 DMCFKFN4   EQU   X'20'    system managed pubset with NK4 volsets
2 *
2 DMCFKUSA   DS    X        volume set usage
2 DMCFKUST   EQU   X'80'    sm pubset with standard volsets
2 DMCFKWRK   EQU   X'40'    sm pubset with work volsets
2 DMCFKHSS   EQU   X'20'    sm pubset with HSMS controlled volsets
2 *
2            DS    XL2      unused
2 DMCFKNOV   DS    F        number of volsets
2            DS    XL1      unused
2 DMCFKMTL   DS    H        maximal I/O transfer length
2 DMCFKUID   DS    CL8      userid allowed to access the pubset
2            DS    XL4      unused
2 *
2 * dynamic allocator values
2 *
2            DS    XL20     unused
2 DMCFKAPA   DS    F        predet primary alloc amount
2 DMCFKASA   DS    F        predet secondary alloc amount
2 DMCFKADL   DS    F        sec alloc doubling limit
2            DS    XL4      unused
2 *
2 * dynamic EAM values
2 *
2 DMCFKEMA   DS    F        maximal size of file SYSEAM
2 DMCFKEMI   DS    F        minimal size of file SYSEAM
2 DMCFKESA   DS    F        secondary allocation of file SYSEAM
2 DMCFKEMS   DS    F        virtual memory size of file SYSEAM
2 *
2 DMCFKREF   DS    XL4      counter of occupations (duplicate)
2 *
```

```
2 DMCFK#     EQU   *-DMCFMST      length of the STAM system
2 *                               managed pubset entry
2            ORG   DMCFD    starting point of dynamic part
2 ***********************************
2 *  dynamic part of a pubset entry  *
2 ***********************************
2 DMCFGOC#   DS    F        counter for occupations
2 *
2 * dynamic pubset status values (1)
2 *
2 DMCFGSTA   DS    X        dynamic status byte (1)
2 DMCFGLOC   EQU   X'80'    set: local          reset: remote
2 DMCFGHOM   EQU   X'40'    set: home           reset: imported
2 DMCFGSH    EQU   X'20'    set: shared         reset: exclusive
2 DMCFGIMC   EQU   X'10'    set: import in process
2 DMCFGEXC   EQU   X'08'    set: export in process
2 DMCFGMAS   EQU   X'04'    set: master         reset: slave
2 DMCFGINA   EQU   X'02'    set: inaccessible   reset: not inacc
2 DMCFGQUI   EQU   X'01'    set: quiet
2 *
2 * dynamic pubset status values (2)
2 *
2 DMCFGST2   DS    X        dynamic status byte (2)
2 DMCFGAC    EQU   X'08'    set: pubset with controlled use
2 DMCFGMCP   EQU   X'04'    set: master change in process
2 DMCFGPAG   EQU   X'02'    set: paging pubset
2 DMCFGERI   EQU   X'01'    set: eram inhibit
2 *
2 DMCFGSES   DS    X        session number
2 *
2 DMCFGFLA   DS    X        CMS flags
2 DMCFGBDF   EQU   X'80'    set:   CMS buffers defined
2 DMCFGBCL   EQU   X'40'    set:   CMS buffers resident (class 3)
2 *                         reset: CMS buffers not resident (class 4)
2 DMCFGBNU   DS    XL2      number of CMS buffers
2 DMCFGATT   DS    X        attribute
2 DMCFGLOB   EQU   X'40'    set: large_objects
2 *                              files/volumes with more than 32 GB
2 DMCFGLFA   EQU   X'20'    set: large_files_allowed
2 DMCFGXCN   DS    CL8      XCS name
2 DMCFGHOS   DS    CL8      host name       :* MSCF host name
2            DS    XL14     unused
2 DMCFGMTL   DS    H        maximal I/O transfer length
2 DMCFGUID   DS    CL8      userid allowed to access pubset
2            DS    XL4      unused
2 *
2 * dynamic allocator values
2 *
```

```
2            DS    XL20     unused
2 DMCFGAPA   DS    F        predet primary alloc amount
2 DMCFGASA   DS    F        predet secondary alloc amount
2 DMCFGADL   DS    F        sec alloc doubling limit
2            DS    XL4      unused
2 *
2 * dynamic EAM values
2 *
2 DMCFGEMA   DS    F        minimal size of file SYSEAM
2 DMCFGEMI   DS    F        maximal size of file SYSEAM
2 DMCFGESA   DS    F        secondary allocation of file SYSEAM
2 DMCFGEMS   DS    F        virtual memory size of file SYSEAM
2 *
2 DMCFGREF   DS    XL4      counter of occupations (duplicate)
2 *
2 DMCFG#     EQU   *-DMCFMST      length of the STAM pubset entry
2 *
2            ORG   DMCFD    starting point of dynamic part
2 ***************************************
2 *  dynamic part of a volume set entry  *
2 ***************************************
2 DMCFEOC#   DS    F        counter for occupations
2 *
2 * dynamic volume set status values (1)
2 *
2 DMCFESTA   DS    X        dynamic status bytes (1)
2 DMCFECON   EQU   X'80'    set: volume set is connected
2 *
2 * dynamic volume set status values (2)
2 *
2 DMCFEST2   DS    X        dynamic status bytes (2)
2 DMCFEMCP   EQU   X'04'    set: master change in process
2 DMCFEERI   EQU   X'01'    set: eram inhibit
2 *
2            DS    XL4      unused
2 *
2 DMCFEATT   DS    X        attribute
2 DMCFERAI   EQU   X'10'    set: volset with RAID volumes
2 DMCFEGSV   EQU   X'08'    set: gs volumes
2 DMCFEDRV   EQU   X'02'    set: high availability by DRV
2 DMCFEKEY   EQU   X'01'    set: key volset
2 *
2            DS    XL16     unused
2 *
2 DMCFEVOL   DS    X        volume set format set
2 DMCFE2KN   EQU   0        NK2 (2K native)
2 DMCFE4KN   EQU   1        NK4 (4K native)
2 DMCFE4KO   EQU   2        NK2, allocation unit multiple of 4K
```

```
2 *                           (4K oriented)
2 * dynamic cache values
2 *
2 DMCFECSZ   DS    F          size of cache buffer
2 *
2 DMCFECB8   DS    X          byte for bit values
2 DMCFECBU   EQU   X'80'      set  : cache size unit is KB
2 *                           reset: cache size unit is MB
2 DMCFECDS   EQU   X'40'      set  : data security ensured
2 DMCFECDB   EQU   X'20'      set  : double recording by buffer
2 DMCFECDD   EQU   X'10'      set  : cache deactivated
2 DMCFECIH   EQU   X'08'      set  : cache in hold
2 DMCFECCU   EQU   X'04'      set  : cache used
2 DMCFECSF   EQU   X'02'      set  : save file failed
2 *
2 DMCFECM    DS    X          cache medium set
2 DMCFECNC   EQU   0          no cache
2 DMCFECDC   EQU   1          controller
2 DMCFECES   EQU   2          expanded storage
2 DMCFECGS   EQU   3          global storage
2 DMCFECMM   EQU   4          main memory
2 *
2 DMCFECS    DS    X          segment size set
2 DMCFEC4    EQU   0          4 KB
2 DMCFEC8    EQU   1          8 KB
2 DMCFEC16   EQU   2          16 KB
2 DMCFEC32   EQU   3          32 KB
2 *
2 DMCFECU1   DS    XL1        GS unit_1
2 DMCFECU2   DS    XL1        GS unit_2
2 *
2 DMCFECFO   DS    X          force out set
2 DMCFECNF   EQU   0          no force out
2 DMCFECIP   EQU   1          at low filling
2 DMCFECIN   EQU   2          at high filling
2 *
2 DMCFECFE   DS    X          prefetch set
2 DMCFECFN   EQU   0          no prefetch
2 DMCFECFL   EQU   1          low
2 DMCFECFH   EQU   2          high
2 *
2 DMCFECFS   DS    X          file selection
2 DMCFEBUS   EQU   0          by user
2 DMCFEALL   EQU   1          all
2 DMCFEAUS   EQU   2          auto select
2 *
2 DMCFECAS   DS    H          size of allocation unit (# of half pages)
2 DMCFEMTL   DS    H          maximal I/O transfer length
```

```
2            DS    XL10     unused
2 *
2 * dynamic allocator values
2 *
2 DMCFEAL5   DS    F        residual space at sat level 5
2 DMCFEAL4   DS    F        residual space at sat level 4
2 DMCFEAL3   DS    F        residual space at sat level 3
2 DMCFEAL2   DS    F        residual space at sat level 2
2 DMCFEAL1   DS    F        residual space at sat level 1
2            DS    XL12     unused
2 DMCFEAZP   DS    F        residual space for ZIP startup
2 *
2            DS    XL16     unused
2 DMCFEREF   DS    XL4      counter of occupations (duplicate)
2 *
2 DMCFE#     EQU   *-DMCFMST      length of the STAM volume
2 *                               set entry
```

DSECT for the output area of the macro (XPAND=OCC)

```
           STAMCE MF=D,PREFIX=D,XPAND=OCC,VERSION=5
1          #INTF REFTYPE=REQUEST,                                        C
1                INTNAME=STAM,                                           C
1                INTCOMP=5
1          MFCHK    MF=D,                                                C
1                SUPPORT=(C,D,E,L,M,S),                                  C
1                PREFIX=D,                                               C
1                MACID=MCH,                                              C
1                DMACID=MCH,                                             C
1                DNAME=MCHOCC,                                           C
1                PARAM=,                                                 C
1                SVC=33,                                                 C
1                ALIGN=F
2 DMCHOCC  DSECT ,
2                *,##### PREFIX=D, MACID=MCH #####
1          STAMLY   MF=D,                                                C
1                PREFIX=D,                                               C
1                MACID=MCH,                                              C
1                PARAM=,                                                 C
1                VERSION=5,                                              C
1                XPAND=OCC,                                              C
1                FUNCT=1,                                                C
1                CG27=DMCH,                                              C
1                CATID=,                                                 C
1                AREA=,                                                  C
1                LENGTH=,                                                C
1                REF=,                                                   C
1                HOST=,                                                  C
1                SELECT=,                                                C
1                PUBSET=
2          #INTF REFTYPE=REQUEST,                                        C
2                INTNAME=STAMLY,                                         C
2                INTCOMP=5
2 DMCHOST    DS    OF
2 DMCHSYS    DS    X        sysid
2 DMCHUNUS   DS    XL3      unused
2 DMCHUSID   DS    CL8      userid
2 DMCHTSN    DS    CL4      tsn
2 DMCHTID    DS    F        tid
2 DMCH#      EQU   *-DMCHOST     length of the occupation entry
```

Example

In the following program, the **STAMCE** macro is called. The catalog ID and the BCAM name of the system of the first 25 MRSCAT entries transferred by **STAMCE** are output on the screen.

```
STAMCE   START
         PRINT NOGEN
         BALR   10,0
         USING  *,10
         USING  DSTAM3,6  --------------------------------------------  (1)
         USING  DSTAM4,7  --------------------------------------------  (2)
         STAMCE MF=E,PARAM=STAM3,VERSION=5  --------------------------  (3)
         LR     7,1  -------------------------------------------------  (4)
         L      6,DMCEAREA  ------------------------------------------  (5)
         L      5,=F'25'
         WROUT  HEADER,WROUTERR -------------------------------------  (6)
SHOW     CLC    DMCFSCTD,=AL4(DMCELAST) ------------------------------  (7)
         BE     WROUTERR
         MVC    CATID,DMCFSCTD ---------------------------------------  (8)
         MVC    PROCESS,DMCFFBCA
         CLI    PROCESS,X'00'
         BNE    WROUT2
         MVC    PROCESS,=CL8' '
WROUT2   WROUT  OUTREC,WROUTERR
         LA     6,DMCFG#(6)
         BCT    5,SHOW
WROUTERR TERM
*** Definitions
STAM3    STAMCE CATID=' ',MF=L,VERSION=5
HEADER   DC     Y(HEADERE-HEADER)
         DC     CL2'  '
         DC     CL1' '
         DC     C'CATID  PROCESSOR '
HEADERE  EQU    *
OUTREC   DC     Y(OUTRECE-OUTREC)
         DC     CL2'  '
         DC     CL1' '
CATID    DS     CL4
         DC     CL3'   '
PROCESS  DS     CL8
OUTRECE  EQU    *
         LTORG
         DS     0F
DSTAM3   STAMCE MF=D,PREFIX=D,XPAND=MCE,VERSION=5
DSTAM4   STAMCE MF=D,PREFIX=D,XPAND=PL,VERSION=5
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,stamce), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,stamce))
%  ASS6011 ASSEMBLY TIME: 210 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 27 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=stamce
%  BLS0523 ELEMENT 'STAMCE', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'STAMCE', VERSION ' ' OF '<date> <time>' LOADED
CATID  PROCESSOR ------------------------------------------------------   (9)
A      N89H04
AAK3   D015B219
AAK4   D015B219
AAN3   D015B219
AA4N
AKEY   HELIOS2
ALB2
ANG3   ANGELA2
AP13   STARTB2
BAB2   BABETTE2
BAB3   BABETTE2
BECK
BEDS   SOPHIE2
BSAD   D015B007
BS41
BUEB   D015B011
BUR3   D017ZE39
BUR4   D017ZE39
BUR5
.
.
.
B202
B203   D015B019
```

(1) Register 6 is assigned to the Assembler as the base address register for addressing the DSECT for the output area of the **STAMCE** macro (in BS2000/OSD-BC V3.0 format). This DSECT is generated at the symbolic address DSTAM3 by means of a **STAMCE** call with MF=D and XPAND=MCE.

(2) Register 7 is assigned to the Assembler as the base address register for addressing the DSECT for the output area of the **STAMCE** macro (in BS2000/OSD-BC V3.0 format). This DSECT is generated at the symbolic address DSTAM4 by means of a **STAMCE** call with MF=D and XPAND=PL.

(3) The **STAMCE** macro is called in its E form. The associated operand list is generated at the symbolic address STAM3 by means of a **STAMCE** call with MF=L, where

CATID=' ' (blank, for all MRSCAT entries)

is entered.
Since no value is specified for AREA, the AREA address is written implicitly to the DMCAREA field of the operand list for the **STAMCE** macro (see DSTAM4: STAMCE MF=D, PREFIX=D, XPAND=PL, VERSION=5).

(4) The start address of the operand list for the **STAMCE** macro is loaded into register R7 so that the output area can be addressed via DMCAREA (points to the address of the output area).

(5) Register R6 is loaded with the address of the output area.

(6) Output of a header line

(7) Check to see whether the end of the MRSCAT entries supplied has been reached.

(8) The first 25 MRSCAT entries that have been transferred to the output area by **STAMCE** are evaluated in a loop: for each entry, the catalog ID (DMCFSCTD field of the DSECT) and the BCAM name of the system (DMCFFBCA field of the DSECT) - or blanks if no BCAM name is entered - are transferred to an output record and output to the screen. The DSECT is subsequently shifted by the length of an entry (DMCFG#).

(9) Output of the first 25 MRSCAT entries (shortened).

STXIT – Specify interrupt event address

General

Application areas:	STXIT processing; see page 131 Starting, interrupting and terminating; see page 72 Communication; see page 163
Macro type:	Type S, MF format **1**: standard/L/E form; see page 29

The following examples of events occurring in the program run affect the continuation of the run:

- invalid operation code, invalid SVC,
- data error, overflow,
- end of program runtime, break interrupts, escape interrupts, or INFORM-PROGRAM command,
- address error, (see table 14 on page 877).

With the following macros, the STXIT procedure enables users to process these types of program interrupts with their own (STXIT) routines and therefore to avoid the danger of premature program termination:

STXIT	Define the STXIT process
EXIT	Terminate the STXIT process
LEVCO	Change the processing level (process priority) of the STXIT process
CONTXT	Enable access to the interrupted task or to the basis process
SETIC	Set timer interval (CPU time or real time)

A STXIT routine is a control section in a main program or a subprogram that is activated as a separate process (own register set (PCB) and own process priority) by the BS2000 Executive in the event of a particular program interrupt. Users respond in the control section to the interrupt event they accepted.

Macro description

Users allocate the STXIT routines of their program to any interrupt events (see table 14 on page 877) with the **STXIT** macro.
The specified allocations are entered internally in a STXIT management block.

Two forms of the **STXIT** macro must be distinguished:

a) **STXIT** macros without the STXDNEW/STXDID operands:

A STXIT management block is created for the first **STXIT** macro. The specified “STXIT event class - STXIT routine” allocations are entered in this management block. Every subsequent macro updates or supplements the allocations entered in this management block (update of the management block). Therefore only one STXIT routine can ever be allocated to an STXIT event class in a program or program system. If an “STXIT event class - STXIT routine” allocation is modified, only the last modification is valid.

b) **STXIT** macro with the STXDNEW/STXDID operands:

A separate STXIT management block with the specified allocations is created for every **STXIT** macro with STXDNEW. The management blocks are chained to each other, and the STXIT routines for the same STXIT event class are activated in the prescribed sequence. An ID for the management block is transferred to the caller. With this ID, the user can supplement, modify or revoke the allocations entered in the relevant management block in a subsequent **STXIT** macro (with STXDID=...).

Both forms of the **STXIT** macro can be used concurrently in one program system. A maximum of 100 STXIT management blocks can be created for one program system.
The processing level of an STXIT process can be changed during execution (**LEVCO** macro).
A STXIT routine is terminated with **EXIT** or **TERM**. The user can specify in the **EXIT** macro whether or not more STXIT routines that are allocated to the STXIT event class are to be started. STXIT processes that are allocated to the same STXIT event class can interrupt each other (nested execution).

Notes

- Only one STXIT routine can be allocated to the “SVC” event class in one and the same program (program system). The allocation must be carried out in the first **STXIT** macro or refer to the first STXIT management block to be created (the SVC number is transferred in register R4 of the STXIT process when the corresponding event occurs).
- The user should note that, in the case of the “CPU timer” or the “real timer”, only one time interval is set at any one moment for the CPU time and/or for the real time (see the last call of the **SETIC** macro).

Macro format and description of operands

STXIT
$\left[\left\{\begin{array}{l}\text{STXDNEW=addr / (r)}\\\text{STXDID=addr / (r)}\end{array}\right\}\right]$,MODE=DEFUNCD / DEFMODE / INTMODE [,STXMSG=addr / (r)] [,TERM=list] [,TIMER=list] [,ERROR=list] [,ABEND=list] [,PROCHK=list] [,RUNOUT=list] [,RTIMER=list] [,ESCPBRK=list] [,HWERROR=list] [,SVC=list] [,SVCLIST=addr / (r) / CLOSE] [,CONTXTL=addr / (r)] [,INTR=list] [,INTRBUF=addr / (r) / CLOSE] ,TERMRUN=STD / FORCED [,MIGRATE=list] [,PARMOD=24 / 31] [,MF=L / (E,..)]

The variable `list` in the operand section corresponds to the following expression:

$$\text{list} \mathrel{\hat{=}} \left\{\begin{array}{l}\text{(addr[,number])}\\\text{((r)[,number])}\\\text{(CLOSE)}\end{array}\right\}$$

where the operand values have the following meanings:

addr
Symbolic address (name) of a STXIT routine.

(r)
Register containing the address value “addr”.

number
Maximum number of STXIT routines in the nesting.
0 ≤ number ≤ 127. Default value: number = 0.

CLOSE
The assignment of STXIT routine to STXIT event class is revoked.

STXDNEW=
A (new) STXIT management block is created. The allocations of STXIT event classes to STXIT routines are entered in the management block. An ID for the management block is transferred to the caller by the system. The caller can use this ID to modify the specified allocations in subsequent **STXIT** macros.

addr
Symbolic address (name) of a field in which the ID is entered; field length = 4 bytes.

(r)
Register containing the address value “addr”.

STXDID=
The allocations specified in a previous **STXIT** macro are to be modified. The relevant STXIT management block is accessed via the returned ID.

addr
Symbolic address (name) of the field with the ID for the management block;
field length = 4 bytes.

(r)
Register containing the address value “addr”.

MODE=
Defines the addressing mode for the STXIT routine. The addressing mode is assigned to the STXIT event classes defined in the same **STXIT** macro (the effect is event-class-specific). By calling several **STXIT** macros with different event classes, the addressing mode can therefore be set separately for each event class.

DEFUNCD
The STXIT routine is started in the same addressing mode that was set at the time of the **STXIT** call. No check is carried out as to whether the same addressing mode is set when the interrupt occurs.

DEFMODE
Default setting: the STXIT routine is started in the addressing mode that was active at the time of the **STXIT** call. However, the STIXIT routine is not activated if the addressing mode at the time of the **STXIT** call is different to the addressing mode at the time the event occurs.

INTMODE
The STXIT routine is started in the addressing mode active at the time of the interrupt.

STXMSG=
Determines the address of a 4 byte field which contains the STXIT message. This message is transferred to the STXIT contingency process (in register R1).

addr
Symbolic address (name) of the message field.

(r)
Register containing the address value addr.

TERM=
Specifies the address of the STXIT routine for the “program termination” STXIT event class (program termination due to synchronous events).

list
See the description of the `list` variable above.

TIMER=
Specifies the address of the STXIT routine for the “CPU timer” STXIT event class.

list
See the description of the `list` variable above.

ERROR=
Specifies the address of the STXIT routine for the “unrecoverable program error” STXIT event class.

list
See the description of the `list` variable above.

ABEND=
Specifies the address of the STXIT routine for the STXIT event class “ABEND” (program termination due to asynchronous events).

list
See the description of the `list` variable above.

PROCHK=
Specifies the address of the STXIT routine for the “program error” event class.

list
See the description of the `list` variable above.

RUNOUT=
Specifies the address of the STXIT routine for the “end of program runtime” STXIT event class.

list
See the description of the `list` variable above.

RTIMER=
Specifies the address of the STXIT routine for the “real timer” STXIT event class.

list
See the description of the `list` variable above.

ESCPBRK=
Specifies the address of the STXIT routine for the “ESCPBRK” (escape break) STXIT event class. In addition to the event code in register R3, the function key code in the rightmost byte of register R4 is transferred to the routine (for function key codes, see the “TIAM” manual [16] or the table in the appendix on page 1153). This functionality is only provided if the TIAM partner is a terminal and not an application (e.g. OMNIS).

list
See the description of the `list` variable above.

HWERROR=
Specifies the address of the STXIT routine for the “hardware error” STXIT event class.

list
See the description of the `list` variable above.

SVC=
Specifies the address of the STXIT routine for the “SVC interrupt” STXIT event class. In addition to the event code in register R3, the SVC number is transferred in the rightmost byte of register R4 to the routine. The monitored SVC is not executed. The interrupted PCB continues to EXIT after the SVC.

list
See the description of the `list` variable above.

SVCLIST=
Specifies the address of a field with SVC numbers. The field must be aligned on a halfword boundary. These SVC numbers are elements of the STXIT event class.
Format:

Byte 0:	number of SVC entries > 0
Byte 1:	SVC number (hexadecimal)
:	:
Byte n:	SVC number (hexadecimal)

addr
Symbolic address (name) of the field containing SVC numbers.

(r)
Register containing the address value “addr”.

CLOSE
The assignment of the STXIT event class “SVC” to the STXIT routine is canceled.

CONTXTL=
Describes the address of a 4-byte field (aligned to word boundary) in which the length of the data exchange field is stored. This is required by the **CONTXT** when used with the operands SAVE and LAYOUT=FCONTXT to exchange data with the PCB of the specified process. There are different lengths for servers with /390 architecture and servers with x86 architecture.

addr
Symbolic address (name) of the length field.

(r)
r = register with the address value addr.

INTR=
Specifies the address of the STXIT routine for the “message for the program” STXIT event class.

list
See the description of the `list` variable above.

INTRBUF=
Specifies the address of a field for a message that is sent with the INFORM-PROGRAM command. The field must be 64 bytes long.
Text length < field length: end of message is marked with X'00'.
Text length > field length: the message is truncated.
Text length = 0: X'00' is entered in the first byte of the field.

addr
Symbolic address (name) of the field for a message.

(r)
Register containing the address value “addr”.

CLOSE
The “INTR-STXIT routine” has not defined a field to receive a message. Any message issued is ignored.

TERMRUN=
Defines the operate mode of the TERM/ABEND STXIT routine.

STD
STXIT routines signed on with STD run according to their placement in the wait queue in line with the LIFO and priority principles.

FORCED
STXIT routines signed on with FORCED always run, and always as last STXIT routines. Their order is determined in line with the LIFO principle.

MIGRATE=
Specifies the address of the STXIT routine for the “Live Migration” STXIT event class.

list
See the description of the `list` variable above.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

STXIT event class, STXIT operand and assigned events

The following table shows:

- which interrupt events an STXIT event class contains,
- which principle (FIFO/LIFO) is used to activate the STXIT routines allocated to an event class (queues, classification),
- the maximum nesting depth that can be specified and
- the event code allocated to an interrupt event.

STXIT event class	STXIT operand	Interrupt result	Event code in R3	Queueing method	Maximum nesting level
Program error	PROCHK	Illegal SVC	X'04'	LIFO	127
		Invalid operationcode	X'58'		
		Data error	X'60'		
		Exponent overflow	X'64'		
		Division error or neg. square root	X'68'		
		Mantissa = 0	X'6C'		
		Exponent underflow	X'70'		
		Decimal overflow	X'74'		
		Fixed-point overflow	X'78'		
CPU timer	TIMER	“SETIC interval” elapsed for CPU time	X'20'	FIFO	127
Real timer	RTIMER	“SETIC interval” elapsed for real time	X'A0'	FIFO	127
		Summer / winter time conversion	X'C0'		
End of program runtime	RUNOUT	CPU time limit for task or program exceeded	X'80'	FIFO	0
Unrecoverable program error	ERROR	Privileged SVC	X'08'	LIFO	127
		Access to a non-existent memory page	X'48'		
		Privileged operation	X'54'		
		Address error (e.g. alignment error, incorrect register)	X'5C'		
		XA error in SVC call (24 bit data area used in 31 bit mode)	X'9C'		

Table 14: STXIT event classes and their associated interrupt events (Teil 1 von 2)

STXIT event class	STXIT operand	Interrupt result	Event code in R3	Queueing method	Maximum nesting level
Unrecoverable program error (cont.)	ERROR	Real timer (condition error)	X'A4'	LIFO	127
		Data area alignment error in SVC call	X'AC'		
		Validation error	X'B0'		
		Invalid UNIT no. in standard header	X'C4'		
Message for the program	INTR	INFORM-PROGRAM command	X'44'	LIFO	127
ESCPBRK	ESCPBRK	BREAK/ESCAPE (via keys)	X'84'	LIFO	127
Program termination due to asynchronous events	ABEND	Error recognized by system, e.g. error in system, power failure	X'88'	LIFO	0
		START-PROGRAM, LOAD-PROGRAM, ABEND, EXIT-JOB CANCEL-JOB	X'8C'		
		Address translation error due to hardware error	X'94'		
		Hardware error (CPU)	X'A8'		
		Forced unloading of a subsystem (system support)	X'B8'		
		Unavoidable DMS error	X'BC'		
Program termination due to synchronous events	TERM	TERM-SVC from a TU program	X'90'	LIFO	0
		Program termination due to CMD/LGOFF macro	X'98'		
SVC interrupt	SVC	Call of a specified SVC	X'50'	LIFO	127
Hardware error	HWERROR	I/O error when using the “data in virtual” method	X'28'	LIFO	0
Live Migration	MIGRATE	Live Migration	X'D0'	FIFO	127

Table 14: STXIT event classes and their associated interrupt events (Teil 2 von 2)

Notes on the macro call

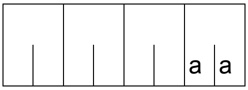
- FIFO = First In First Out; LIFO = Last In First Out.
- ESCPBRK operand: this STXIT event affects only program interrupts via function keys. It does not include program interrupts resulting from a HOLD-PROCEDURE or ESCAPE, or HOLD-PROGRAM or BREAK command.
- INTRBUF operand: where the nesting depth is greater than 0, the message text is overwritten by a subsequent INFORM-PROGRAM command.
- SVC and SVCLST operands: if SVC 128 (STXIT group) is specified as an interrupt event, it can lead to a loop.
- SVC and SVCLST operands: the monitored SVC is not executed and the interrupted PCB continues to EXIT after the SVC. This can also occur with the TERM operand.
- Invalid address in the **STXIT** macro: the “address error” event is reported, and an STXIT routine allocated in a previous **STXIT** macro is activated or the program is terminated with “address error”.
- If the **TERM** macro is called in a program system, it causes all the STXIT routines allocated to the STXIT event class TERM to be activated. Another **TERM** macro (also in an STXIT routine - instead of **EXIT**) causes immediate program termination, if required.
- A STXIT routine of the ABEND or TERM event class can be tested with the interactive debugging aid AID.
  Exception: STXIT routine that was activated via a CANCEL-JOB command.

Return information and error flags

Register R3 of the STXIT process contains the interrupt weight (event code).

If an interrupt event of the ESCPBRK class occurs, the function key code is transferred in register R4 of the STXIT process (see the “TIAM” manual [16] or the Appendix on page 1153 for details of function key codes).
This functionality is only provided if the TIAM partner is a terminal and not an application (e.g. OMNIS).
If an interrupt event of the SVC class occurs, the SVC number is transferred in register R4 of the STXIT process.

R15:



A return code relating to the execution of the STXIT macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Normal execution.
X'04'	Function was not executed. Invalid operands.
X'08'	Function was not executed. Invalid ID for the STXIT control block (STXDID operand).
X'0C'	Function was not executed. No memory space for the STXIT control block.
X'10'	Function was not executed. A maximum of 100 STXIT control blocks can be created for a program system.
X'14'	Function was not executed. The STXIT event class SVC is only valid for the first STXIT management block to have been created.
X'1C'	Function was executed. STXIT control block cannot be updated because the memory key in the PCB does not match the memory key in the control block.

SUSPEND – Suspend task

General

Application area:	Contingency processing; see page 110 STXIT processing; see page 131 Event control; see page 94
Macro type:	Type S, MF format **1**: standard/E/L form; see page 29

Macro description

The **SUSPEND** macro places the calling basic task or contingency process in a wait state until a (STXIT) contingency process starts.
Caution: If no (STXIT) contingency process is started, the process issuing **SUSPEND** remains in the wait state.

If **SUSPEND** is used in conjunction with event control (e.g. asynchronous **SOLSIG** followed by **SUSPEND** macro), the user is responsible for ensuring that there is also a **POSSIG** signal so that the wait state can be terminated with the aid of the contingency process thus started.

Macro format and description of operands

SUSPEND
[MF=L / E]

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

Return information and error flags

R15:

b	b					a	a

A structured return code (aa=primary return code, bb=secondary return code) relating to the execution of the SUSPEND macro is transferred in register R15.

X'aa'	X'bb'	Meaning
X'00'	X'00'	Calling task suspended.
X'04'	X'04'	Macro permitted only in TU programs. No action.
X'10'	X'04'	Invalid operands entered. No action.

SWITCH – Set and query job and user switches

General

Application areas:	User and job switches; see page 73 Communication; see page 163
Macro type:	Type S, MF format **3**: C/D/L/E/M form; see page 29

- The **SWITCH** macro combines the functionality of the GETSW, GETUS, SETSW and SETUS macros.

32 **user switches** are available to each user ID. These switches are stored in the user catalog. Only the user switches in the user catalog of the home pubset are used.
The user switches are numbered consecutively from 0 through 31. When a user ID is set up, all 32 of its switches are off. Thereafter they maintain whatever setting the user assigns to them.
Each switch can be activated, deactivated or inverted individually. User switches are permanent switches, i.e. they maintain their setting even after EXIT-JOB.

The operating system provides 32 **job switches** for each job. These job switches are numbered consecutively from 0 through 31 and stored in TCB lists. Unlike user switches, job switches are always switched off at the start of a job. Users must decide themselves what the setting of each switch implies for their own programs.
Each switch can be activated, deactivated or inverted individually. Job switches are temporary switches, i.e. they maintain their setting only until termination of the job (EXIT-JOB). Note that job switches may also be used by some system components and utility routines (see section “User and job switches” on page 73). When the SET-JOB-STEP command is executed, switches 16 through 31 are turned off.

Macro description

The **SWITCH** macro enables users to activate, deactivate, invert and query the user switches assigned to their user ID and the job switches assigned to their jobs.

Macro format and description of operands

SWITCH
[MODE=TASK / USER] ,USERID=*OWN / addr [,SWITCH=addr / (no, ...)] ,ACTION=*READ / *WRITE / *ON / *OFF / *INVERT / addr ,CONST=YES / NO ,MF=D / C / L / E / M [,PARAM=addr / (r)] ,PREFIX=J / p ,MACID=CSS / macid

The operands are described below in alphabetical order.

ACTION=
Specifies what function is to be executed.

***READ**
Default setting: all user or job switches are queried. The SWITCH operand is not evaluated.
Information indicating which switches are on or off is stored in a 4-byte field of the parameter list (default: JCSSSW) following the MF=E call. In this field, bit 2^n corresponds to switch n. If bit 2^n is set to “1”, switch n is on. If it is set to “0”, the switch is off.

***WRITE**
All switches specified in the SWITCH operand are activated. Switches that are not specified are deactivated.

***ON**
All switches specified in the SWITCH operand are activated. Switches that are not specified remain unchanged.

***OFF**
All switches specified in the SWITCH operand are deactivated. Switches that are not specified remain unchanged.

***INVERT**
All switches specified in the SWITCH operand are inverted. Switches that are not specified remain unchanged.

addr
Symbolic address (name) of a 1-byte field containing the function to be executed in the following form:

```
0  Query switches
1  Activate/deactivate switches
2  Activate switches
3  Deactivate switches
4  Invert switches
```

This operand value is not permitted in conjunction with MF=L.

CONST=
Specifies whether or not equates are to be generated.

YES
Default setting: equates are generated.

NO
No equates are generated.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

In the E form of the macro, the address of the operand list is stored in the PARAM operand. Default setting: JCSS$PL

MODE=
Specifies whether a given function is to be executed for user switches or job switches. This operand is mandatory unless MF=M is specified.

TASK
Job switches are queried or set.
The USERID operand is not evaluated.

USER
The user switches assigned to the specified user ID are queried or set.

SWITCH=
Specifies the user switches or job switches whose setting is to be queried or modified.
If the SWITCH operand is not specified, the preset bit mask X'00000000' is used.

(no, ...)
no = switch number (0 .. 31).
This operand value lists the numbers of the switches whose setting is to be queried or modified, or specifies a subset of switches to be modified if ACTION=*WRITE.

The parentheses must be used even if only one switch is specified.

addr
Symbolic address (name) of a 4-byte field (bit mask) in which each bit corresponds to a user or job switch as follows:

bit 2^0 ≙ switch 0, bit 2^1 ≙ switch 1, ..., bit 2^{31} ≙ switch 31

This bit mask specifies which switches are to be modified. The type of modification is determined by the ACTION operand.
If a bit is set to “1”, the corresponding switch is activated, deactivated or inverted. If the bit is set to “0”, the switch is not modified unless ACTION=*WRITE: in this case, the switches whose bit is set to “0” are deactivated.

This operand value is not permitted in conjunction with MF=L.

USERID=
Specifies the user ID whose user switches are to be queried or modified.
This operand is not evaluated for job switches (MODE=TASK).

<u>*OWN</u>
Default setting: the user switches of the user's own ID are to be queried or modified.

addr
Symbolic address (name) of an 8-byte field containing the desired user ID as a string. This string is left-justified (padded, if necessary, with trailing blanks).

*If ACTION≠*READ, another user's ID can be specified only under the privileged user ID TSOS or if the caller has the “USER-ADMINISTRATION” system privilege.*
*If ACTION=*READ, no special privileges are necessary to query the switches of another user's ID.*

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the SWITCH macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Function executed successfully.
X'02'	X'00'	X'0001'	Warning: the function was executed but an internal error occurred involving permanent user switches. This return code can only occur for user switches (MODE=USER).
X'00'	X'01'	X'0002'	Operand error.
X'00'	X'40'	X'0008'	The specified user ID does not exist.
X'00'	X'82'	X'000C'	The specified user ID is locked.
X'00'	X'82'	X'0010'	The required access authorization is missing.
	X'20'	X'0020'	(Various) internal errors.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Layout of the DSECT for MODE=TASK

```
           SWITCH MF=D,MODE=TASK
1 *--------------------- START OF SWITCH -----------------------------*
1          MFCHK MF=D,PREFIX=J,MACID=CSS,PARAM=,                         C
1                SVC=42,                                                 C
1                DMACID=CSS,SUPPORT=(D,L,C,M,E)
2 JCSS     DSECT ,
2                *,##### PREFIX=J, MACID=CSS #####
1 *
1          #INTF REFTYPE=REQUEST,INTNAME=SWITCH,INTCOMP=001
1 JCSS$PL  DS    0F          BEGIN OF PARAMETERAREA
1          FHDR  MF=(C,JCSS),EQUATES=YES
2          DS    0A
2 JCSSFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 JCSSIFID DS    0A              0   INTERFACE IDENTIFIER
2 JCSSFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 JCSSFCT  DS    AL1             2   FUNCTION NUMBER
2 JCSSFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 JCSSRET  DS    0A              4   GENERAL RETURN CODE
2 *
2 * GENERAL_RETURN_CODE CLEARED (X'00000000') MEANS
2 * REQUEST SUCCESSFUL PROCESSED AND NO ADDITIONAL INFORMATION
2 *
2 JCSSSRET DS    0AL2            4   SUB RETURN CODE
2 JCSSSR2  DS    AL1             4   SUB RETURN CODE 2
2 * ALWAYS CLEARED (X'00') IF MAIN_RETURN_CODE IS X'FFFF'
2 * Standard subcode2 values as defined by convention:
2 JCSSR2OK EQU   X'00'               All correct, no additional info
2 JCSSR2NA EQU   X'01'               Successful, no action was necessary
2 JCSSR2WA EQU   X'02'               Warning, particular situation
2 JCSSSR1  DS    AL1             5   SUB RETURN CODE 1
2 *
2 * GENERAL INDICATION OF ERROR CLASSES
2 *
2 * CLASS A    X'00'           FUNCTION WAS SUCCESSFULLY PROCESSED
2 * CLASS B    X'01' - X'1F'   PARAMETER SYNTAX ERROR
2 * CLASS C    X'20'           INTERNAL ERROR IN CALLED FUNCTION
2 * CLASS D    X'40' - X'7F'   NO CLASS SPECIFIC REACTION POSSIBLE
2 * CLASS E    X'80' - X'82'   WAIT AND RETRY
2 *
2 JCSSRFSP EQU   X'00'               FUNCTION SUCCESSFULLY PROCESSED
```

```
2 JCSSRPER EQU   X'01'               PARAMETER SYNTAX ERROR
2 *  3 GLOBALLY DEFINED ISL ERROR CODES IN CLASS X'01' - X'1F'
2 JCSSRFNS EQU   X'01'               CALLED FUNCTION NOT SUPPORTED
2 JCSSRFNA EQU   X'02'               CALLED FUNCTION NOT AVAILABLE
2 JCSSRVNA EQU   X'03'               INTERFACE VERSION NOT SUPPORTED
2 *
2 JCSSRAER EQU   X'04'               ALIGNMENT ERROR
2 JCSSRIER EQU   X'20'               INTERNAL ERROR
2 JCSSRCAR EQU   X'40'               CORRECT AND RETRY
2 *  2 GLOBALLY DEFINED ISL ERROR CODES IN CLASS X'40' - X'7F'
2 JCSSRECR EQU   X'41'               SUBSYSTEM (SS) MUST BE CREATED
2 *                                  EXPLICITELY BY CREATE-SS
2 JCSSRECN EQU   X'42'               SS MUST BE EXPLICITELY CONNECTED
2 *
2 JCSSRWAR EQU   X'80'               WAIT FOR A SHORT TIME AND RETRY
2 JCSSRWLR EQU   X'81'                    "     LONG       "
2 JCSSRWUR EQU   X'82'               WAIT TIME IS UNCALCULABLY LONG
2 *                                  BUT RETRY IS POSSIBLE
2 *  2 GLOBALLY DEFINED ISL ERROR CODES IN CLASS X'80' - X'82'
2 JCSSRTNA EQU   X'81'               SS TEMPORARILY NOT AVAILABLE
2 JCSSRDH  EQU   X'82'               SS IN DELETE / HOLD
2 *
2 JCSSMRET DS    0AL2            6   MAIN RETURN CODE
2 JCSSMR2  DS    AL1             6   MAIN RETURN CODE 2
2 JCSSMR1  DS    AL1             7   MAIN RETURN CODE 1
2 *
2 * SPECIAL LAYOUT OF LINKAGE_MAIN_RETURN_CODE (YYYY IN X'00XXYYYY')
2 *
2 JCSSRLNK EQU   X'FFFF'             LINKAGE ERROR / REQ. NOT PROCESSED
2 JCSSFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1 *
1 * ****************************************************************
1 * *** END OF STANDARD HEADER - START SPECIAL SWITCH PARAMETERLIST
1 * ****************************************************************
1 *
1 JCSSHDR  EQU   X'009A0B01',4  std header task switch (TU)
1 *
1 *
1 *           *****  SET OF RETURN CODES  *****
1 *   OUT OF THE SYSTEM-WIDE DEFINED RETURN-CODES, THE FOLLOWING MAY
1 *   BE EXPECTED (CONFER INCLUDE FHDRI):
1 *      00 01 FFFF       SPECIFIED FUNCTION IS NOT SUPPORTED
1 *      00 03 FFFF       SPECIFIED VERSION IS NOT SUPPORTED
1 *      00 04 FFFF       ALIGNMENT ERROR
1 *
1 *   ADDITIONAL SPECIAL RETURNCODES ARE DEFINED :
1 *
```

```
1 *      00 00 0000       NORMAL EXECUTION
1 *      02 00 0001       EXECUTION, BUT ERROR IN WHENQ PROCESSING
1 *      00 01 0002       PARAMETER ERROR
1 *      00 40 0008       USERID NOT FOUND
1 *      00 82 000C       USERID SEVERED
1 *      00 82 0010       NO PRIVILEGED
1 *      XX 20 0020       SYSTEM ERROR
1 *
1 *   MAIN RETURNCODES
1 JCSSOK   EQU   X'0000'   execution ok
1 JCSSWHQE EQU   X'0001'   execution with warning
1 JCSSPARE EQU   X'0002'   parameter error
1 JCSSUNFE EQU   X'0008'   userid not found
1 JCSSUSEE EQU   X'000C'   userid severed
1 JCSSNPRE EQU   X'0010'   no privileged
1 JCSSIERR EQU   X'0020'   internal error
1 *
1 *
1 *   DATA AREA
1 JCSSACT  DS    XL1         ACTION
1 JCSSREA  EQU   0              = *READ
1 JCSSWRT  EQU   1              = *WRITE
1 JCSSON   EQU   2              = *ON
1 JCSSOFF  EQU   3              = *OFF
1 JCSSINV  EQU   4              = *INVERT
1 *
1 JCSSRES  DS    XL3         FILLER
1 *
1 JCSSSW   DS    F           SWITCHES 31-0
1 *
1          ORG   JCSSSW
1 JCSSSW3  DS    XL1         SWITCHES 31 - 24
1 JCSSS31  EQU   X'80'          = SWITCH 31
1 JCSSS30  EQU   X'40'          = SWITCH 30
1 JCSSS29  EQU   X'20'          = SWITCH 29
1 JCSSS28  EQU   X'10'          = SWITCH 28
1 JCSSS27  EQU   X'08'          = SWITCH 27
1 JCSSS26  EQU   X'04'          = SWITCH 26
1 JCSSS25  EQU   X'02'          = SWITCH 25
1 JCSSS24  EQU   X'01'          = SWITCH 24
1 *
1 JCSSSW2  DS    XL1         SWITCHES 23 - 16
1 JCSSS23  EQU   X'80'          = SWITCH 23
1 JCSSS22  EQU   X'40'          = SWITCH 22
1 JCSSS21  EQU   X'20'          = SWITCH 21
1 JCSSS20  EQU   X'10'          = SWITCH 20
1 JCSSS19  EQU   X'08'          = SWITCH 19
1 JCSSS18  EQU   X'04'          = SWITCH 18
```

```
1 JCSSS17  EQU   X'02'           = SWITCH 17
1 JCSSS16  EQU   X'01'           = SWITCH 16
1 *
1 JCSSSW1  DS    XL1          SWITCHES 15 -  8
1 JCSSS15  EQU   X'80'           = SWITCH 15
1 JCSSS14  EQU   X'40'           = SWITCH 14
1 JCSSS13  EQU   X'20'           = SWITCH 13
1 JCSSS12  EQU   X'10'           = SWITCH 12
1 JCSSS11  EQU   X'08'           = SWITCH 11
1 JCSSS10  EQU   X'04'           = SWITCH 10
1 JCSSS9   EQU   X'02'           = SWITCH  9
1 JCSSS8   EQU   X'01'           = SWITCH  8
1 *
1 JCSSSW0  DS    XL1          SWITCHES  7 -  0
1 JCSSS7   EQU   X'80'           = SWITCH  7
1 JCSSS6   EQU   X'40'           = SWITCH  6
1 JCSSS5   EQU   X'20'           = SWITCH  5
1 JCSSS4   EQU   X'10'           = SWITCH  4
1 JCSSS3   EQU   X'08'           = SWITCH  3
1 JCSSS2   EQU   X'04'           = SWITCH  2
1 JCSSS1   EQU   X'02'           = SWITCH  1
1 JCSSS0   EQU   X'01'           = SWITCH  0
1 *
1 JCSSUID  DS    CL8          USERID
1 *
1 JCSS#    EQU   *-JCSS$PL    LENGTH OF PARAMETERAREA
1 *--------------------- END OF SWITCH -------------------------------*
```

Example

```
SWITCH   START
         PRINT NOGEN
SWITCH   AMODE ANY
         BALR  R3,0
         USING *,R3
         MVC   TASKSW,INITSW
         SWITCH MF=M,ACTION=*READ
         SWITCH MF=E,MODE=TASK,PARAM=TASKSW                               (2)
SW1      CLC   JCSSMRET,=Y(JCSSOK)
         BNE   ERROR
*
         TM    JCSSSW0,JCSSS1                                              (3)
SW2      BZ    END
         SWITCH MF=M,ACTION=*INVERT,SWITCH=(2,3)
         SWITCH MF=E,MODE=TASK,PARAM=TASKSW
         SWITCH MF=M,ACTION=*READ
         SWITCH MF=E,MODE=TASK,PARAM=TASKSW                               (3)
SW3      CLC   JCSSMRET,=Y(JCSSOK)
         BNE   ERROR
         B     END
*
ERROR    EQU   *
******   ... ERROR HANDLING ... **************
         B     END
END      TERM
R3       EQU   3
INITSW   SWITCH MF=L,MODE=TASK
TASKSW   SWITCH MF=C,MODE=TASK
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,switch), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,switch)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 395 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 84 MSEC
//end
%  ASS6012 END OF ASSEMBH
```

```
/mod-job-sw on=(1,2,3,4,5) ——————————————————————————————————————————————  (1)
/load-program *m(macexmp.lib,switch),test-options=*aid,run-mod=*adv
/%in sw1
/%in sw2
/%in sw3
/%r
STOPPED AT LABEL: SW1 , SRC_REF: 20, SOURCE: SWITCH , PROC: SWITCH
/%d jcsssw %x;%r ————————————————————————————————————————————————————————  (2)
*** TID: 009301BB *** TSN: 6WWQ *******************************************
CURRENT PC: 00000012    CSECT: SWITCH  ***********************************
V‘00000090‘ = JCSSSW   + #‘00000000‘
00000090 (00000000) 0000003E                                      ....
STOPPED AT LABEL: SW2 , SRC_REF: 24, SOURCE: SWITCH , PROC: SWITCH
/%d jcsssw0 %x;%r ———————————————————————————————————————————————————————  (3)
CURRENT PC: 00000020    CSECT: SWITCH  ***********************************
V‘00000093‘ = JCSSSW0  + #‘00000000‘
00000093 (00000000) 3E                                            .
STOPPED AT LABEL: SW3 , SRC_REF: 52, SOURCE: SWITCH , PROC: SWITCH
/%d jcsssw %x;%r ————————————————————————————————————————————————————————  (4)
CURRENT PC: 0000003E    CSECT: SWITCH  ***********************************
V‘00000090‘ = JCSSSW   + #‘00000000‘
00000090 (00000000) 00000032                                      ....
```

(1) Prior to program execution, switches 1 through 5 are activated for demonstration purposes.

(2) The job switches are read and the settings output to the field JCSSSW: X'0000003E' = $2^5 + 2^4 + 2^3 + 2^2 + 2^1$ means that switches 1, 2, 3, 4 and 5 are activated and all other switches are deactivated.

(3) A query is issued as to whether switch 1 is activated: The field JCSSSW0 contains the value X'3E'. The result of the logical comparison with the bitmap is not zero. Processing of the program is continued.

(4) Following inversion of switches 2 and 3, all switches are read and their settings output to the field JCSSSW: X'00000032' = $2^5 + 2^4 + 2^1$ means that switches 1, 4 and 5 are activated and all other switches are deactivated.

SYSFL – Reassign system files

General

Application area:	System files; see page 156
Macro type:	Type S, MF format **1**: standard/E/L/C/D form; see page 29

The (standard) file names SYSDTA, SYSLST, SYSLST01 ... SYSLST99 and SYSOUT denote files used by the operating system to input commands and data to the operating system or to output data via the operating system. These files are each created by the task and specify input and output areas that were preset (primarily) from the start.
Users can revoke the primary assignment and assign their own (cataloged) files to the (standard) file names. Some of the standard names can also be equated. The file which is being assigned (to the right of the equals sign) then takes over the functions of the (system) file (to the left of the equal sign). The **SYSTA** macro can be used to output the current assignment of system files.

The system files available for input/output are described in the general section “System files” on page 156.

Macro description

The **SYSFL** macro allows the user to issue the appropriate command via the macro interface of the Macro Command Language Processor (MCLP) without interrupting program mode (see section “Macro Command Language Processor macros” on page 45).

The **SYSFL** macro enables the user to reassign the (system) files SYSDTA, SYSLST, SYSLST01,..., SYSLST99 and SYSOUT.
Output device and output format specifications for the (system) file SYSLST can be entered by the user.
The **SYSFL** macro also enables the user to specify an object module file (TASKLIB) for the dynamic binder loader.
Messages concerning the execution of the macro are output to SYSOUT and, if desired, may be copied into an area in the calling program.

Macro formats and description of operands

The following table shows the various macro formats for the **SYSFL** macro. The individual formats and their operands are then described.

SYSFL
[,PARMOD=24 / 31]
[,MF=D / (E,..) / C]

For a description of the PARMOD and MF operands, see section “S-type macros” on page 29.

Assignment of **SYSDTA** (see description on page 898):

SYSFL
'SYSDTA=pathname / #filename / (SYSCMD) / (PRIMARY)' 'SYSDTA=pathname1(element), VERSION=*STD / vers, TYPE=*STD / type'
[,adr / (r)]
[,PARMOD=24 / 31]
[,MF=L]

Assignment of **SYSOUT** (see description on page 900):

SYSFL
'SYSOUT=pathname / (pathname,EXTEND) / *DUMMY / (PRIMARY)'
[,addr / (r)]
[,PARMOD=24 / 31]
[,MF=L]

Assignment of **SYSLST** (see description on page 901):

SYSFL
'SYSLST=pathname / (pathname,EXTEND) / #filename / (SYSCMD) / (PRIMARY)' [,addr / (r)] [,PARMOD=24 / 31] [,MF=L]

Output of **SYSLST** to printer (see description on page 903):

SYSFL
'FILE=SYSLST, PRINTER=136 / 160 [,HREC=m] [,FORM=code] [,LOOP=vfb]] [,COPIES=number1 / ([number1],number2)] [,CHARS=(c1[,c2][,c3][,c4])] ,CONTROL=NO / PHYS [,IMAGE=xxxx] [,SHIFT=columns] [,DIS=zz]' [,addr / (r)] [,PARMOD=24 / 31] [,MF=L]

Assignment of **SYSLSTn** (see description on page 904):

SYSFL
'SYSLSTn=pathname / (pathname,EXTEND) / *DUMMY / (PRIMARY) / *SYSLSTn' [,addr / (r)] [,PARMOD=24 / 31] [,MF=L]

Assignment for dynamic binder loader **DBL** (see description on page 906):

SYSFL
'TASKLIB=pathname / (NO)' [,addr / (r)] [,PARMOD=24 / 31] [,MF=L]

Notes

- The operands must be enclosed in single quotes.
- Unlike system output files, the assigned cataloged files are not automatically output to the printer. The user can have these files printed using the PRINT-DOCUMENT command.
- If the parentheses are omitted in the (SYSCMD) or (PRIMARY) entry, these will be interpreted as file names.
- The field to receive the SYSOUT log must be aligned on a word boundary. The output record starts with the 4-byte record length field, which contains the record length in bytes 0-1. The record length field is followed by the output text. Structure of the output area:

```
         DS   0F
OUTPUT   DC   Y(AENDE-OUTPUT)   Output area
SLF      DS   0CL4              4 bytes: record length field
SL       DS   CL2               2 bytes: record length of output record
         DS   CL2               2 bytes: reserved
TEXT     DS   CL300             Output text
AENDE    EQU *
```

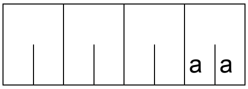
Notes on temporary files:

- The system can operate without temporary files.
- Temporary files are task-oriented and are deleted on task termination.
- Temporary files can be of the type BTAM, SAM, ISAM or PAM.

Return information and error flags

Messages concerning command processing are part of the SYSOUT log, which can optionally be transferred to the program (“addr” operand).

R15:



A return code relating to the execution of the SYSFL macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Normal termination.
X'04'	Insufficient memory; request not processed.
X'08'	Operand list error (address area).
X'0C'	Last output record placed in user area was truncated.
X'10'	Macro/command error (command returned an error to MCLP).

Assignment of SYSDTA

```
SYSFL

'SYSDTA=pathname / #filename / (SYSCMD) / (PRIMARY)'
'SYSDTA=pathname1(element), VERSION=*STD / vers, TYPE=*STD / type'

[,adr / (r)]
[,PARMOD=24 / 31]
[,MF=L]
```

SYSDTA=

pathname
"pathname" stands for: [:catid:][$userid.]filename

catid
Catalog ID of the pubset where the file is stored.
The catalog ID assigned to the user ID in the user catalog.

userid
User ID to which the file is assigned.
Default value: user ID from the SET-LOGON-PARAMETERS command.

filename
Name of a cataloged file (or a file generation).
The file must be a SAM or ISAM file with variable record length.
An ISAM file also requires a key starting in column 5 and a key length of 8 bytes (KEYPOS=5, KEYLEN=8; see the "DMS Macro" manual [7]). Specification of a file group (not to be confused with a file generation group) is permitted only for tape files.

#filename
Name of a temporary file.
# = character that is set by the system parameter TEMPFILE (see the "Introduction to Systems Suppport" manual [10]) as a prefix to the file names of temporary files. Information on the character selected can be obtained with the SHOW-SYSTEM-PARAMETERS command or the macro NSIOPT.
filename = any file name; length ≤ 30 characters.
See *Notes on temporary files* on page 897.

(SYSCMD)
The system file SYSCMD is equated with the system file SYSDTA, i.e. SYSCMD assumes the same function as SYSDTA in addition to its own function. Data (and not only commands) can then be read from the SYSCMD system file.

(PRIMARY)
SYSDTA is restored to its original primary assignment.

SYSDTA=pathname1(element)
“pathname1” stands for [:catid:][$userid.]library

catid	see SYSDTA=pathname.
userid	see SYSDTA=pathname.
library	name of a PLAM library. The expression “library(element)” may be up to 41 characters long.
element	name of a library element. The following characters are permissible: – alphabetic charactersA,...Z – special characters$, #, -, @ – numeric characters0,...9 The first character must be alphabetic. The last character must not be a hyphen.

VERSION=
The element name is supplemented by the version specification.

*STD
Default setting: the highest version is used.

vers
Version designation (max. 10 characters).

TYPE=
Element type (1 letter).

*STD
Default setting: element type = S.

type
Character from the set (D, S, M).

addr
Symbolic address (name) of a field to receive the SYSOUT log; for structure see page 897.

(r)
Register containing the address value “addr”.

Assignment of SYSOUT

SYSFL
'SYSOUT=pathname / (pathname,EXTEND) / *DUMMY / (PRIMARY)' [,addr / (r)] [,PARMOD=24 / 31] [,MF=L]

SYSOUT=

pathname
“pathname” stands for [:catid:][$userid.]filename

catid
Catalog ID of the pubset where the file is stored.
The catalog ID assigned to the user ID in the user catalog.

userid
User ID to which the file is assigned.
Default value: user ID from the SET-LOGON-PARAMETERS command.

filename
Name of a file or a file generation.
The file is created as a SAM file on a public volume, with its size being determined by the system parameter SSMAPRI and the SECONDARY-ALLOCATION being determined by the system parameter SSMASEC. Alternatively, this file may also reside on private volumes, if so defined earlier by the user in a CREATE-FILE command. However, no multifile tape may be used.

It is advisable to estimate the probable size of the file “filename” and to specify a corresponding PRIMARY-ALLOCATION operand in the CREATE-FILE command so as to avoid too many memory requests.

(pathname,EXTEND)
SYSOUT is assigned to the file “filename”; data records are entered from the end of the file onwards.

***DUMMY**
A dummy file is assigned to SYSOUT. For meaning see ADD-FILE-LINK command, “Commands” manual [19]. Data records are not stored.

(PRIMARY)
Restores the SYSOUT file to its primary assignment.

addr
Symbolic address (name) of a field to receive the SYSOUT log; for structure see page 897.

(r)
Register containing the address value “addr”.

Assignment of SYSLST

SYSFL
'SYSLST=pathname / (pathname,EXTEND) / #filename / (SYSCMD) / (PRIMARY)'
[,addr / (r)]
[,PARMOD=24 / 31]
[,MF=L]

SYSLST=

pathname
“pathname” stands for: [:catid:][$userid.]filename

catid
Catalog ID of the pubset on which the file is stored.
The catalog ID assigned to the user ID in the user catalog.

userid
User ID that is assigned to the file.
Default value: user ID from the SET-LOGON-PARAMETERS command.

filename
Name of a file or a file generation. SYSLST is assigned to this file or file generation. The file is created as a SAM file on a public volume, with its size being determined by the system parameter SSMAPRI and the SECONDARY-ALLOCATION being determined by the system parameter SSMASEC. Alternatively, this file may also reside on private volumes, if so defined earlier by the user in a CREATE-FILE command. However, no multifile tape may be used.

It is advisable to estimate the probable size of the file “filename” and to specify a corresponding PRIMARY-ALLOCATION operand in the CREATE-FILE command so as to avoid too many memory requests.

If no further memory space is available during SYSLST output to a disk file, the system requests a tape. The file is automatically copied to this tape and deleted from the disk. SYSLST output then continues to the tape file.

#filename
Name of a temporary file.
# = character that is set by the system parameter TEMPFILE (see the “Introduction to Systems Suppport” manual [10]) as a prefix to the file names of temporary files. Information on the character selected can be obtained with the SHOW-SYSTEM-PARAMETERS command or the macro NSIOPT.
filename = any file name; length ≤ 30 characters.
See *Notes on temporary files* on page 897.

***DUMMY**
A dummy file is assigned to SYSLST (for an explanation, see the ADD-FILE-LINK command in the “Commands” manual [19]); data records are not stored.

(pathname,EXTEND)
The “filename” file is assigned to SYSLST; the data records are entered from the end of the file onwards.

(PRIMARY)
Restores the SYSLST file to its primary assignment.

addr
Symbolic address (name) of a field to receive the SYSOUT log; for structure see page 897.

(r)
Register containing the address value “addr”.

Output of SYSLST to printer

SYSFL
'FILE=SYSLST, PRINTER=136 / 160 [,HREC=m] [,FORM=code] [,LOOP=vfb]] [,COPIES=number1 / ([number1],number2)] [,CHARS=(c1[,c2][,c3][,c4])] ,CONTROL=NO / PHYS [,IMAGE=xxxx] [,SHIFT=columns] [,DIS=zz]' [1] [,addr / (r)] [,PARMOD=24 / 31] [,MF=L]

[1] The PRINTER, HREC, COPIES, CHARS, CONTROL, IMAGE, SHIFT and DIA operands refer to the PRNT macro of SPOOL V2.7 and are no longer described in the current manuals.

FILE=SYSLST
Together with the operands below, specifies the output format of the (system) file SYSLST.

addr
Symbolic address (name) of a field to receive the SYSOUT log; for structure, see page 897.

(r)
Register containing the address value “addr”.

The FORM and LOOP operands describe the printout format; they are described under the **PRNTDOC** macro (see “SPOOL & Print - Macros and Exits” manual [23]).

Assignment of SYSLSTn

SYSFL
'SYSLSTn=pathname / (pathname,EXTEND) / *DUMMY / (PRIMARY) / *SYSLSTm' [,addr / (r)] [,PARMOD=24 / 31] [,MF=L]

SYSLSTn=
n = a (2-digit) number from the range (01,02,...,99).
The SYSLSTn files can only be used if (cataloged) SAM files are assigned to them. The primary assignment to these files is the file that is simultaneously assigned to the SYSLST (system) file.

pathname
pathname stands for: [:catid:][$userid.]filename

catid
Catalog ID of the pubset on which the file is stored.
The catalog ID assigned to the user ID in the user catalog.

userid
User ID to which the file is assigned.
The user ID from the SET-LOGON-PARAMETERS command.

filename
Name of a file or a file generation. The file is assigned to SYSLSTn and is created as a SAM file on public volumes.
It is advisable to estimate the probable size of the “filename” file and to specify an appropriate PRIMARY-ALLOCATION operand in the CREATE-FILE command.

***DUMMY**
SYSLSTn is assigned a dummy file. The data records are not stored.

(pathname,EXTEND)
SYSLST is assigned the “filename” file; data records are entered from the end of the file onwards.

(PRIMARY)
Primary assignment.

***SYSLSTm**
m = a 2-digit number from the range (01,02,...,99); m ≠ n
The SYSLSTn (system) files can also be assigned to each other. The following points should be borne in mind:

- mutual assignment is not permitted

  Example of incorrect assignment

  ```
  SYSFL     SYSLSTn  = *SYSLSTm
  SYSFL     SYSLSTm  = *SYSLSTn
  ```

- the assignment must eventually lead to a cataloged file or to a dummy file

  Example

  ```
  SYSFL     SYSLSTn  = filename
  SYSFL     SYSLSTm  = *SYSLSTn
  SYSFL     SYSLSTp  = *SYSLSTm
  ```

addr
Symbolic address (name) of a field to receive the SYSOUT log; for structure, see page 897.

(r)
Register containing the address value “addr”.

Assignment for dynamic binder loader DBL

SYSFL
'TASKLIB=pathname / (NO)' [,addr / (r)] [,PARMOD=24 / 31] [,MF=L]

TASKLIB=
Denotes an object module file which is to be searched by the binder loader DBL if

- no object module file (library) was specified on loading the program and/or
- external references still have to be resolved.

TASKLIB=(PRIMARY), i.e. the (user) file TASKLIB or,if this does not exist, the file $TSOS.TASKLIB is searched.

pathname
“pathname” stands for: [:catid:][$userid.]filename

catid
Catalog ID of the pubset on which the file is stored.
The catalog ID assigned to the user ID in the user catalog.

userid
User ID to which the file is assigned.
User ID from the SET-LOGON-PARAMETERS command.

filename
Name of an object module file.
When a program is linked, this object module file is searched before the TASKLIB (user) file, or the (system) file $TSOS.TASKLIB, for the object module required by the binder loader (DBL).

Notes

- “filename” must not be the name of a file generation.
- Procedure nesting: After the END-PROCEDURE or EXIT-PROCEDURE commands, the TASKLIB assignment valid before the procedure call applies.

(NO)
The assignment is canceled. TASKLIB=(PRIMARY) applies, or, in the case of nested procedures, the assignment that existed before the procedure call is retained.

addr
Symbolic address (name) of a field to receive the SYSOUT log; for structure see page 897.

(r)
Register containing the address value “addr”.

SYSTA – Output information on system file and TASKLIB assignment

General

Application area:	System files; see page 156
Macro type:	Type S, MF format **1**: standard/E/L/C/D form; see page 29

The (standard) file names SYSDTA, SYSCMD, SYSLST, SYSLST01, SYSLST02,..., SYSLST99 and SYSOUT denote files used by the operating system to input data and commands to the operating system or to output data via the operating system. These files are created by the appropriate task and specify input and output areas that were preset from the start.
The **SYSFL** macro allows users to revoke the primary assignment and assign their own (cataloged) files to the (standard) file names. Some of the standard names can also be equated.

Macro description

The **SYSTA** macro enables the user to issue the SHOW-SYSTEM-FILE-ASSIGNMENTS command via the macro interface of the macro command language processor (MCLP) without interrupting program mode (see section “Macro Command Language Processor macros” on page 45).
Messages concerning command processing are issued on SYSOUT and are also entered in a program area of the calling program. The SHOW-SYSTEM-FILE-ASSIGNMENTS command permits the user to receive information on the assignment of system files and the object module file (TASKLIB) for the dynamic binder loader.

Macro format and description of operands

SYSTA
'([SYSCMD][,SYSDTA][,SYSOUT][,SYSLST][,SYSLST01][,SYSLST02]...[,SYSLST99][,TASKLIB])' [,addr / (r)] ,SYSOUT=YES / NO ,DIALOG=NO / YES [,PARMOD=24 / 31] [,MF=L / (E,..) / D / C]

SYSCMD / SYSDTA / SYSOUT / SYSLST / SYSLST01 / SYSLST02 / .. / SYSLST99 / TASKLIB

Several system files, including TASKLIB, may be specified. Parentheses can be omitted if only one file is specified.If the operands are omitted, the user receives information about all system files and the TASKLIB. These operands must be enclosed in single quotes.

addr

Symbolic address (name) of a field to receive the SYSOUT log.

(r)

Register containing the address value “addr”.
This field must be aligned on a word boundary. The output record starts with the 4-byte record length field, which contains the record length in bytes 0-1. The record length field is followed by the output text.
Structure of the output area:

```
         DS  0F
OUTPUT   DC  Y(AENDE-OUTPUT)   Output area
SLF      DS  0CL4              4 bytes: record length field
SL       DS  CL2               2 bytes: record length
                         DS  CL2               2 bytes: reserved
TEXT     DS  CL300             Output text
AENDE    EQU *
```

SYSOUT=

Specifies whether the log is also to be output to SYSOUT.

YES

Default setting: the log is also output to SYSOUT.

NO

The log is not output to SYSOUT.
In this case, the “addr” of the receiving field must be specified.

DIALOG

Specifies whether an error dialog is to be conducted if syntax errors are detected.

NO

Default setting: no error dialog is conducted.

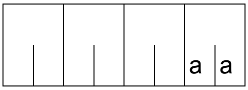
YES

An error dialog is to be conducted.

Return information and error flags

Messages concerning command processing are part of the SYSOUT log, which is also transferred to the output area (“addr” operand).

R15:



A return code relating to the execution of the SYSTA macro up to command execution is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	Normal termination.
X'04'	Insufficient memory area is available; the request was not carried out.
X'08'	Error in the operand list (address area).
X'0C'	The last output record entered in the user area is truncated.
X'10'	Macro/command error (the command returned an error to the MCLP).

TCHNG – Modify terminal characteristics

Application area: Data terminal communication; see page 160
Macro type: Type O; see page 28

- This macro description applies to TIAM V13.2A.

Macro description

The **TCHNG** macro enables characteristics of the logical terminal to be modified by the user program.
The modifications made by the **TCHNG** macro remain effective until termination of the user program or until a further **TCHNG** macro is issued; they are valid only for this program's I/O operations for the terminal with the **RDATA**, **WROUT** and **WRTRD** macros.

Macro format and description of operands

```
TCHNG
─────────────────────────────────────────────────────────────────────
 ⎧ EDOPT=DYN                                                        ⎫
 ⎪                                                                  ⎪
 ⎪              ⎧ ,MODE=LINE ,OHCOPY= ⎧ N ⎫ ,OHOM= ⎧ N ⎫          ⎫ ⎪
 ⎪              ⎪                     ⎩ Y ⎭       ⎩ Y ⎭          ⎪ ⎪
 ⎪              ⎪                                                ⎪ ⎪
 ⎪              ⎪ ,OINFO= ⎧ N ⎫ ,ONOPOSN= ⎧ N ⎫                   ⎪ ⎪
 ⎪              ⎪         ⎩ Y ⎭           ⎩ Y ⎭                   ⎪ ⎪
 ⎪              ⎪                                                ⎪ ⎪
 ⎪              ⎪ ,OBELL= ⎧ N ⎫ ,IGETBS= ⎧ N ⎫                    ⎪ ⎪
 ⎨              ⎪         ⎩ Y ⎭          ⎩ Y ⎭                    ⎪ ⎬
 ⎪ EDOPT=STAT   ⎨                                                ⎬ ⎪
 ⎪              ⎪ ,ILCASE= ⎧ N ⎫ ,IGETFC= ⎧ N ⎫                   ⎪ ⎪
 ⎪              ⎪          ⎩ Y ⎭          ⎩ Y ⎭                   ⎪ ⎪
 ⎪              ⎪                                                ⎪ ⎪
 ⎪              ⎪ ,IGETIC= ⎧ N ⎫ ,ICFD= ⎧ N ⎫                     ⎪ ⎪
 ⎪              ⎪          ⎩ Y ⎭        ⎩ Y ⎭                     ⎪ ⎪
 ⎪              ⎪                                                ⎪ ⎪
 ⎪              ⎪ ,MODE=FORM ,IGETBS= ⎧ N ⎫ ,ILCASE= ⎧ Y ⎫        ⎪ ⎪
 ⎩              ⎩                     ⎩ Y ⎭         ⎩ N ⎭        ⎭ ⎭
,OFLOW=SYS / USER
,SUB=OUT / OUTIN
,INFOLIN=NO / YES
,CLEAR=YES / NO
```

The operands are described below in alphabetical order.

CLEAR=
Determines whether the screen is to be cleared when the output mode changes.

<u>YES</u>
Default setting: When the output mode changes, e.g.
from `LINE` to `FORM`
from `FORM` to `LINE, PHYS, COMP`
from `PHYS` to `LINE, FORM, COMP`
from `COMP` to `FORM`

the screen is to be cleared. If the change of mode does not occur after an input, the system waits t seconds before clearing the screen to allow users to read their last output (see the command MODIFY-TERMINAL-OPTIONS OVERFLOW-CONTROL=...).

NO
The screen is not cleared when the output mode changes. The following message is output without a waiting period.

Note

The user program must ensure that the screen contents do not affect any function of the new mode, e.g. by premodified fields.

EDOPT=
Specifies which edit options values are to be evaluated.

<u>DYN</u>
Default setting: edit options values specified in any subsequent **RDATA**,**WROUT** and **WRTRD** macro calls are to be evaluated (default values set by the system).

STAT
The edit options values specified in the **TCHNG** macro for MODE, OBELL, OHCOPY, OHOM, OINFO, ONOPOSN, IGETBS, ILCASE, IGETFC, IGETIC and ICFD are to be evaluated in any subsequent **RDATA**, **WROUT** and **WRTRD** macro calls. These specifications remain valid until modified by a further **TCHNG** macro or until end of program. However, they have no influence on system inputs and outputs. EDOPT=STAT is ignored if **VTSUCB** is used.

INFOLIN=
Determines how incoming informative messages are to be displayed at data terminals.

NO
Default setting: Incoming informative messages are to be displayed like normal line-mode messages at data terminals without a system line.

YES
At data terminals whose hardware does not provide a system line, incoming informative messages are to be displayed in the last line on the screen when a formatted or physical message is currently displayed on the screen (see the **WROUT** macro).

OFLOW=
Specifies the type of overflow check.

SYS
Default setting: In the event of long user program outputs, the system is to perform an overflow check to prevent information overflow at the terminal. The type of system overflow check is specified by the terminal user via the MODIFY-TERMINAL-OPTIONS command (see the “Commands” manual [19], for default values and effect).

USER
The system is not to take any precautions to prevent information overflow in the event of long user program outputs. An overflow check can thus be performed individually by the user program.

Note

Operating system messages are still subject to the overflow check preset by the system or specified via the MODIFY-TERMINAL-OPTIONS command.

SUB=
Determines when the defined substitution character is to replace characters.

OUT
Default setting: The defined substitution character is to replace illegal characters for line mode outputs.

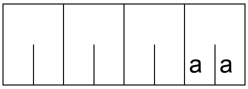
OUTIN
The defined substitution character is to

- replace illegal characters for line mode outputs
- transfer the logical control character SUB to the user program for line mode inputs (see the **VTCSET** macro).

MODE=
OHCOPY=
OHOM=
OBELL=
ONOPOSN=
OINFO=
IGETBS=
ILCASE=
IGETFC=
IGETIC=
ICFD=
For the meaning of these operands, see the **WRTRD** or **WROUT** macro.

Return information and error flags

R15:



A return code relating to the execution of the TCHNG macro is transferred in the rightmost byte of register R15.

X'aa'	**Meaning**
X'00'	Normal termination.
X'04'	Unrecoverable error.
X'08'	Operand error.
X'0C'	The caller is not in timesharing mode.
X'10'	Operand error.
X'14'	Invalid edit option.

TERM – Terminate program and procedure step

General

Application areas:	Starting, interrupting and terminating; see page 72 Debugging aids; see page 162
Macro type:	Type S, MF format **1**: standard/E/L form; see page 29

Macro description

The **TERM** macro performs the following functions:

- Terminate program (default value)
- Terminate program and procedure step (UNIT=STEP operand)
- Issue memory dump (DUMP operand)
- Transfer return code to program-monitoring job variable (URETCD operand)

All input/output operations initiated by the program before the execution of the macro are completed prior to program termination.

Macro format and description of operands

TERM
UNIT=PRGR / STEP ,DUMP=N / Y ,MODE=**N**ORMAL / **A**BNORMAL [,URETCD=code / addr / (r)] ,MF=S / (E,..) / L

UNIT=
Determines whether or not a distinction is to be made between modes during program termination.

PRGR
Default setting: the program is terminated.

STEP
Terminates the program, taking into account the mode in which the program was executing.

- Interactive mode:
  If the program was called in a non-S procedure, the system also branches to the next SET-JOB-STEP, EXIT-JOB, END-PROCEDURE or CANCEL-PROCEDURE command. If the program was called in an S procedure, the system also protects SDF-P error handling.
- Batch mode (ENTER file):
  The system also branches to the next SET-JOB-STEP or EXIT-JOB command.

Note
The following specifications are recommended:
UNIT=PRGR if MODE=NORMAL
UNIT=STEP if MODE=ABNORMAL

DUMP=
Determines whether a memory dump is to be issued.

N
Default setting: no memory dump is issued.

Y
Issues a memory dump unless DUMP=NO was specified in the MODIFY-TEST-OPTIONS command.

MODE=
Specifies the way in which the program is to terminate.

NORMAL
Default setting: the program is to terminate normally.
For users of job variables: The status indicator of a program-monitoring variable, if present, is set to C'$T␣'.

ABNORMAL
The program is to terminate abnormally. The message
`.... ABNORMAL PROGRAM TERMINATION (&00)` is output.

`(&00)= NRT0001` if UNIT=PRGR was specified
`(&00)= NRT0101` if UNIT=STEP was specified

For users of job variables:
The status indicator of a program-monitoring job variable, if present, is set to C'$A␣'.
See also the *Note* above.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

The following operand is only available to users who have the software product JV (see the “JV” [22] manual) at their disposal:

URETCD=code
Specifies a 1- to 4-byte alphanumeric value in decimal (C'cccc') or hexadecimal (X'xxxxxxxx') form. The program returns this value as a return code to the program-monitoring job variable (left-justified, bytes 4-7).

Note

> If the operand is missing, the value C'␣␣␣␣' is returned to the program-monitoring job variable.
> This operand is ignored if no program-monitoring job variable has been defined.
>
> **addr**
> Relative virtual address of a 4-character alphanumeric value. The program returns this value as a return code to the program-monitoring job variable (bytes 4-7).
>
> **(r)**
> Register containing a 4-character alphanumeric value. This value is returned to the program-monitoring job variable as a return code (bytes 4-7). Register R0 is destroyed if URETCD=(r) is specified.

Functional description

When this macro is executed the following occurs:

- All files assigned to the program and still open are closed.
- Memory assigned to the program is deallocated.
- Any STXIT routine defined for the event class TERM is activated.
- AIDSYS is called with the event “TERM”.
- Bytes 8-30 are deleted in the physical device table for each released device. The first byte in the operation list is set to X'FF'. The program start address is set to 0 (4 bytes) in the program table entry.
- The system then changes over to command mode.

Notes on the macro call

- The operand is ignored if an invalid address is specified in the URETCD operand.
- Register R1 contains the operand list address.

  If the operand list address is invalid or invalid operands were specified,
  TERM UNIT=STEP,MODE=ABNORMAL,DUMP=Y
  is executed and the following error message is issued:
  `%.... ABNORMAL PROGRAM TERMINATION NRT0601`
- When the **TERM** macro with the operand DUMP=Y is called, the message
  `PROCESSING INTERRUPTED AT... appears.`
  Output of the memory dump depends on the value of the DUMP operand specified in the MODIFY-TEST-OPTIONS command. **TERM** activates one of the following messages in the case of DUMP=STD (default value):
  - in interactive mode:
    `DUMP DESIRED ? REPLY (Y=YES, N=NO)`
    Whether or not a dump is produced depends on the response to the query.
  - in batch mode and in procedures:
    `SYSTEM REGULATIONS PROHIBIT DUMP`
    No dump is produced.

  The dump is output to disk in unedited form as a PAM file (see the **CDUMP2** macro). The file containing the dump is created under the user ID of the user who requested it. As soon as the dump is complete, the message
  `DUMP WRITTEN,FILENAME=$userid.DUMP.tsn.i`
  and the title line of the dump are output.
  If two or more dumps are requested for the same TSN, they are numbered consecutively (“i”). The file can be evaluated with the aid of the analysis program DAMP.

TINF – Read or modify task attributes

General

Application area:	Starting, interrupting and terminating; see page 72
Macro type:	Type S, MF format **1**: standard/E/L form; see page 29

Macro description

By means of the **TINF** macro the user can read or modify

- the run priority of the task
- the task attribute
- the parameters for deactivation prohibition on the task

in accordance with the values laid down in the user catalog.

It is also possible for the calling task to join or to leave an affinity task group.
Tasks which frequently require write access to the same data are said to be affined to one another. This common data can be located either in the system address space (which is used by all tasks) or in the user address space within common memory pools. Such tasks can be grouped together to form an affinity task group.
However, a task can only ever be assigned to a single task group (or to no task group). The functionality of affined task groups is implemented by the TANGRAM subsystem (see also the “Introduction to System Administration” [10]).

Macro format and description of operands

TINF
ACCESS=R / W [,DEACT=Y / N] [,DWTR=Y / N] [,DSSR=Y / N] [,TPRYAD=addr / (r)] [,TTYPAD=addr / (r)] [,TGAFF=Y / N,TGIDAD=addr / (r)] [,PROCNAD=addr / (r)] [,PARMOD=24 / 31] [,MF=L / (E,..)]

ACCESS=
Controls the read/write function of the operands TPRYAD and TTYPAD.

R
The data is transferred to the specified fields.

W
The data is transferred from the specified fields to the TCB.

DEACT=
Specifies whether or not a task can be deactivated in the following cases:

- the utilization of system capacity (CPU, memory, paging rate) is very high.
- the task assumes particular wait states or remains in these (see the DWTR operand). If the DWTR operand is also specified, its setting has priority.
- because of the system services it has used so far the task is available for deactivation (see the DSSR operand). If the DSSR operand is also specified, its setting has priority.

Y
The task can be deactivated.

N
The task is not to be deactivated. It can still be deactivated for other reasons (e.g. VPASS macro with a wait time of over 500 ms).

DWTR=
Specifies whether a task can be deactivated in the following cases:

- It assumes particular, longer-lasting wait states, e.g. because of a PASS.
- It remains longer in an active wait state, e.g. after a SOLSIG with COND=UNCOD or after a MSG7X/TYPIO with REPLY.

Y
The task may be deactivated.

N
The task is not to be deactivated.

DSSR=
Specifies whether a task can be deactivated after having used a certain amount of system services (e.g. CPU time).

Y
The task may be deactivated.

N
The task is not to be deactivated.

TPRYAD=

Indicates the run priority of the task.

addr

Address of a one-byte field containing the run priority value.
For ACCESS=R the run priority of the task is transferred from the TCB into the specified field. For ACCESS=W the run priority is transferred from the specified field into the TCB.

(r)

Register containing the address value “addr”.

TTYPAD=

Specifies the task attribute.
Values for the task attribute:

```
TTYPTP   EQU X'81'   transaction job
TTYPIACT EQU X'40'   interactive job
TTYPPB   EQU X'20'   batch job
```

With ACCESS=R the job type is transferred to the specified field. With ACCESS=W the task attribute is transferred from the specified field to the TCB.

addr

Address of a one-byte field containing the value for the task attribute.

(r)

Register containing the address value “addr”.

TGAFF=

Specifies whether the calling task wishes to join or leave an affinity task group.
This operand must be specified in conjunction with TGIDAD.

Y

The task joins the task group whose task group ID is in the field addressed by TGIDAD or, if the field contains zero, a new task group. If a new task group is set up, the new task group ID is returned in the field addressed by TGIDAD.

N

The task leaves the group whose task group ID is in the field addressed by TGIDAD. If the task leaving is the last in its task group, that task group is deleted and the task group ID is set to zero in the field addressed by TGIDAD.

TGIDAD=
Designates the task group ID address for the calling task.
This operand may only be specified in conjunction with TGAFF.

addr
Symbolic address of a 4 byte field containing the task group ID.

(r)
Register containing the address value of addr.

PROCNAD=
Provides the following information:

- Is the calling task assigned to an affinity task group?
- If so, how many CPUs are currently assigned to this task?
- How many CPUs is the system currently using?

addr
Symbolic address of a 4 byte field.
The first 2 bytes contain the number of CPUs on which the task may currently run.
The second 2 bytes contain the total number of CPUs currently available.
If the calling task is not assigned to a task group or if the TANGRAM subsystem is not loaded, both sub-fields contain the same figures.

(r)
Register containing the address value of addr.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Notes for affinity task groups

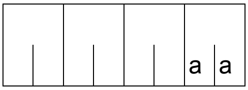
– When a task joins a task group, a memory pool must already have been set up so that the field designated with TGIDAD can be placed in an area accessible to all the tasks in the application. This field must be initialized with zero. The first task then sets up the group on joining, and all the other tasks just join the existing group.

– BS2000 takes over when it comes to synchronizing joining procedures.
The second task to join (chronologically speaking) is only processed when the first has been successfully joined and the new group ID has been returned.

Return information and error flags

During macro processing, register R1 contains the operand list address.

R15:



A return code relating to the execution of the TINF macro is transferred in the rightmost byte of register R15.

X'aa'	Meaning
X'00'	The macro was executed successfully.
X'04'	Invalid operands have been specified. At least one required function has not been executed.
X'08'	The field designated with TGIDAD does not contain a valid group ID for the task.
X'10'	For at least one function there is no authorization.
X'0C'	The task wishing to join a task group already belongs to another task group.
X'14'	The task wishing to leave (TGAFF=N operand) does not belong to a group.
X'18' [1]	Up to 65535 task groups can be joined. This limit has been exceeded.
X'1C' [1]	Internal TANGRAM error. Function has not been fully executed.
X'20' [1]	Internal TANGRAM administration function (TANGBAS subsystem) unavailable.

[1] When this return code occurs, applications (e.g. database retrieval) can be continued

Example

The **TINF** macro transfers run priority, task attribute and information to an area in the user program:

```
TINF     START
         PRINT NOGEN
TINF     AMODE ANY
         BALR  3,0
         USING *,3
CHKOUT1  TINF  TPRYAD=PRI,TTYPAD=TYPE,PROCNAD=INFO,                  *
               TGAFF=N,TGIDAD=AFFINI,PARMOD=31 ———————————————————————  (1)
CHKIN    TINF  TGAFF=Y,TGIDAD=AFFINI,PARMOD=31 ———————————————————————  (2)
CHKOUT2  TINF  TGAFF=N,TGIDAD=FALSE,PARMOD=31 ————————————————————————  (3)
CHKOUT3  TINF  TGAFF=N,TGIDAD=AFFINI,PARMOD=31 ———————————————————————  (4)
END      TERM
*
****  Definitions  ****
PRI      DS    L1
TYPE     DS    L1
INFO     DS    F
AFFINI   DC    X'00000000'
FALSE    DC    X'08150815'
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,tinf), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,tinf)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 283 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 79 MSEC
//end
%  ASS6012 END OF ASSEMBH
/load-executable-program library=macexmp.lib,element-or-symbol=tinf, -
/     test-options=*aid
%  BLS0523 ELEMENT 'TINF', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'TINF', VERSION ' ' OF '<date> <time>' LOADED
/%in chkin
/%in chkout2
/%in chkout3
/%in end
/%r
STOPPED AT LABEL: CHKIN , SRC_REF: 40, SOURCE: TINF , PROC: TINF
/%d %15 %xl, pri %xl1, type %xl1, affini %xl;%r ─────────────────────── (1)
*** TID: 00AF0265 *** TSN: 6WPP *******************************************
CURRENT PC: 00000024    CSECT: TINF  *************************************
%15             = 00000014   ....
V‘0000007A‘ = PRI      + #‘00000000‘
0000007A (00000000) D2                                     K
V‘0000007B‘ = TYPE     + #‘00000000‘
0000007B (00000000) 40
V‘00000080‘ = AFFINI   + #‘00000000‘
00000080 (00000000) 00000000                               ....
STOPPED AT LABEL: CHKOUT2 , SRC_REF: 59, SOURCE: TINF , PROC: TINF
/%d %15 %xl, affini %xl;%r ──────────────────────────────────────────── (2)
CURRENT PC: 00000038    CSECT: TINF  *************************************
%15             = 00000000   ....
V‘00000080‘ = AFFINI   + #‘00000000‘
00000080 (00000000) E3C70021                               TG..
STOPPED AT LABEL: CHKOUT3 , SRC_REF: 78, SOURCE: TINF , PROC: TINF
/%d %15 %xl, false %xl;%r ───────────────────────────────────────────── (3)
CURRENT PC: 0000004C    CSECT: TINF  *************************************
%15             = 00000008   ....
V‘00000084‘ = FALSE    + #‘00000000‘
```

```
00000084 (00000000) 08150815                                 ....
STOPPED AT LABEL: END , SRC_REF: 96, SOURCE: TINF , PROC: TINF
/%d %15 %xl, affini %xl;%r ——————————————————————————————————————————— (4)
CURRENT PC: 00000060    CSECT: TINF  ***************************************
%15            = 00000000   ....
V‘00000080‘ = AFFINI   + #‘00000000‘
00000080 (00000000) 00000000                                 ....
```

(1) The task should be logged off from the affinity task group with the task group ID (X'00000000') in the `AFFINI` field:
Register 15 contains the return code X'00000014', i.e. the task to be logged off did not form part of a task group.
The run priority of the task is X'D2' = 210.
The task is an interactive task (X'40').
The contents of the field `AFFINI` are unchanged.

(2) The task logs on to a newly set up affinity task group (field `AFFINI` contains the input X'00000000').
Register 15 contains the return code X'00000000', i.e. the **TINF** was executed without errors.
The task group ID of the newly set up affinity task group is returned in the field `AFFINI`. Field `AFFINI` now contains X'E3C70021'.

(3) The task is to be logged off from the affinity task group with the task group ID (X'08150815') located in the `FALSE` field:
Register 15 contains the return code X'00000008', i.e. in the field designated with TGIDAD there is no valid task group ID for this task.
The contents of the field `FALSE` are unchanged.

(4) The task is to be logged off from the affinity task group with the task group ID (X'E3C70021') in the `AFFINI` field:
Register 15 contains the return code X'00000000', i.e. the **TINF** was executed without errors.
Since the task was the last (and only) one in its task group, the task group was deleted and the task group ID was set to Null in the field addressed by TGIDAD.
Field `AFFINI` again contains X'00000000'.

TMODE – Interrogate job attributes

General

Application areas:	Requesting and accessing lists and tables; see page 155 Communication; see page 163
Macro type:	Type O; see page 28

See the **DTMODE** macro for a description of the output fields. The symbolic field names used when the 24-bit interface is generated start with the prefix TSK instead of TMOD.

Macro description

The **TMODE** macro provides the user program with information about the job under which it is running. The user program supplies an area in which this information is to be stored. The macro format depends on the interface desired.

Note

> The user ID under which the user program is running can also be interrogated with the **RDUID** macro. The privilege of the job under which the user program is running can also be interrogated with the **CHKPRV** macro.

Macro formats and descriptions of operands

Format 1: 24-bit interface call

```
[name] TMODE

[ { area[,length] } ]
  { D             }

[,PARMOD=24 / 31]
```

name

Name of the DSECT if operand D is specified.
Default value: name = TSKINF0.

area

Name of an area in which the job information is to be stored. “D” must not be used as the name.

length
Specifies the size in bytes of “area”. If the length operand is omitted, the length attribute of “area” is used. If, in either case, the length is less than the number of bytes of information supplied, the data is truncated by the difference.
The length of the information supplied is available as symbolic constant L@TSKINF (EQU value).

D
A dummy section (DSECT) is generated for the output list.

PARMOD=
Controls macro expansion. If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

no entry
The operand values may also be supplied in an operand list, the address of which is contained in register R1.
The operand list has the following format:

Byte	Operand
0	Must be set to X' 00' ; this is important, since it is the only way of guaranteeing successful execution of the macro. If this entry is omitted, an invalid return code might be transferred.
1 - 3	Address of the area in which job information is to be stored.
4 - 5	Length of the area (here the symbolic constant L@TSKINF may be specified).

Format 2: 31-bit interface call

TMODE
[PARMOD=31] [,PARLIST=addr / (r)]

PARMOD=
Controls macro expansion.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

PARLIST=
Gives the address of an area where the job data is to be stored. The area should be aligned on a word boundary and must start with the standard header.
The **DTMODE** macro generates a description (DSECT/data section) for the I/O area; the initialization values for the standard header are entered.

addr
Symbolic address (name) of the area.

(r)
Register containing the address value “addr”.

Return information and error flags

R15:

0	0	a	a	a	a	a	a

A return code relating to the execution of the TMODE macro is transferred in register R15.

X'aaaaaa'	Meaning
X'000000'	Function has been executed.
X'000004'	Operand error.
X'00000C'	System error.
X'01FFFF'	Incorrect specification for UNIT/FUNCTION in the standard header.
X'03FFFF'	Incorrect specification for VERSION in the standard header.

Example

A number of job attributes (type of data display terminal, size of terminal buffer, TSN of the job, user ID, ...) are to be interrogated. The macro is executed in 31-bit addressing mode. The output area is generated by means of the **DTMODE** macro (with the standard header being initialized). Output fields are edited as required for output by means of the **WROUT** macro.

```
  TMODE    START
           PRINT NOGEN
  TMODE    AMODE ANY
           BALR  3,0
           USING *,3
           TMODE PARMOD=31,PARLIST=TMODPL  ------------------------------  (1)
           UNPK  HFIELD(3),TMODTYPE(2)
           MVC   TASKT+2(2),HFIELD
           TR    TASKT+2(2),CODTAB
           UNPK  HFIELD(5),TMODBUFS(3)
           MVC   TASKB+2(4),HFIELD
           TR    TASKB+2(4),CODTAB
           UNPK  HFIELD(3),TMODPRI(2)
           MVC   TASKP+2(2),HFIELD
           TR    TASKP+2(2),CODTAB
           MVC   TSN,TMODTSN
           MVC   USERID,TMODUSER
           MVC   ACC,TMODACCT
           MVC   JOB,TMODNAME
           WROUT AUSB,TERM,MODE=LINE,PARMOD=31  -------------------------  (2)
2                *,@DCEO      999    921011   53531004
  TERM     TERM
           DS    0F
```

```
AUSB     DC    Y(AUSBE-AUSB)
         DS    CL3
         DC    X'15'
         DC    C'TERMINAL TYPE: '
TASKT    DC    C'X''00'''
         DC    X'15'
         DC    C'BUFFER TERMINAL: '
TASKB    DC    C'X''0000'''
         DC    X'15'
         DC    C'RUN PRIORITY: '
TASKP    DC    C'X''00'''
         DC    X'15'
         DC    C'TSN: '
TSN      DS    CL4
         DC    X'15'
         DC    C'USER ID: '
USERID   DS    CL8
         DC    X'15'
         DC    C'ACCOUNT NUMBER: '
ACC      DS    CL8
         DC    X'15'
         DC    C'JOB NAME: '
JOB      DS    CL8
         DC    X'15'
AUSBE    EQU   *
HFIELD   DS    CL5
CODTAB   DS    CL240
         DC    C'0123456789'
         DC    C'ABCDEF'
         DS    0F
         DTMODE DSECT=NO ——————————————————————————————————————————— (3)
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,tmode), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,tmode))
%  ASS6011 ASSEMBLY TIME: 336 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 86 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=tmode
%  BLS0523 ELEMENT 'TMODE', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'TMODE', VERSION ' ' OF '<date> <time>' LOADED
TERMINAL TYPE: X'35' —————————————————————————————————————————————————  (4)
BUFFER TERMINAL: X'07F8'
RUN PRIORITY: X'D2'
TSN: 2QSE
USER ID: QM212
ACCOUNT NUMBER: 89002
JOB NAME: MACTEST
```

(1) The **TMODE** macro is called. The 31-bit interface is generated; the output area is generated by means of the **DTMODE** macro.

(2) The interrogated values are output in LINE mode by means of **WROUT**. Some values had to be edited before output (unpacked and converted into printable characters).

(3) The output area is generated by means of the **DTMODE** macro. The standard header is initialized.

(4) Output in LINE mode:
terminal type = X'35'.
buffer size = 2040 bytes.
run priority = 210.
etc.

TSPRIO – Output run priorities

General

Application area: Requesting and accessing lists and tables; see page 155
Macro type: Type O; see page 28

Macro description

The **TSPRIO** macro outputs the minimum and maximum values for fixed and variable priorities.

Macro format

TSPRIO

```
          PRINT  GEN
          TSPRIO
EPRIFIXU  EQU     30 Maximum constant priority
EPRIFIXL  EQU    127 Minimum constant priority
EPRIVARU  EQU    128 Maximum variable priority
EPRIVARL  EQU    255 Minimum variable priority
```

TSTAT – Interrogate terminal attributes

General

Application areas: Data terminal communication; see page 160
Requesting and accessing lists and tables; see page 155

Macro type: Type S, MF format **1**:
31-bit interface: standard/L/E/C/D form; see page 29

- This macro description applies to TIAM V13.2A.

Macro description

Information about a terminal may be requested in timesharing mode by means of the **TSTAT** macro. The information obtained refers to the generated device type.

Macro format and description of operands

TSTAT
TCHAR / PHDIM / LIDIM / **VDT**YP / EDOPT / OFLOW / STNAM / PRNAM / ALL / MONCS / PERPH / BASIC ,area [,length] [,MF=C / (C,pre) / (E,..) / (D,pre) / D / L]

TCHAR
Requests physical type of terminal.
Supplies the type under which the terminal was generated in PDN.

PHDIM
Requests physical dimensions of terminal (line mode).

LIDIM
Requests logical dimensions of terminal (line mode).

VDTYP
Requests logical type of terminal.

EDOPT
Requests static edit options.

OFLOW
Requests type of overflow control.

STNAM
Requests terminal name.

PRNAM
Requests processor name.

ALL
Requests output of all information.

MONCS
Requests information on the monitor and the character sets of the terminal.

PERPH
Requests information on the connected peripherals.

BASIC
Requests basic information on the terminal.

area
Symbolic address of an area in which the information requested is stored. The area must be aligned on a halfword boundary.

length
Specifies the length of the area.

- for ALL : 64 bytes
- for MONCS : at least 14 bytes
- for BASIC : 640 bytes
- for other options : 8 bytes

If this specification is missing, the length attribute “area” is used. If the ALL, MONCS or BASIC operand is specified and the area length is smaller than the terminal information supplied, the amount of information which can be supplied is limited to that which will fit in this area.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A prefix (pre = 1..3 letters) can be specified in the C form and D form of the macro, as shown in the macro format.
Default value: pre = TST

The operands TCHAR, PHIDIM, LIDIM, VDTYP, EDOPT, OFLOW, STNAM, PRNAM ALL, MONCS, PERPH and BASIC correspond to those described for the **DCSTA** macro.

Functional description

The calling program may define the area which is to receive the information either itself or by means of the **DCSTA** C,... call (see Example 2). If the program defines the destination area itself, it may perform addressing with the aid of a DSECT generated by means of the **DCSTA** D,... call.

For a description of information supplied see the **DCSTA** macro (page 381).

Return information and error flags

R15:

0	0	0	0	0	0	a	a

A return code relating to the execution of the TSTAT macro is transferred in register R15.

X'aa'	Meaning
X'00'	Normal termination.
X'04'	Unrecoverable error.
X'08'	Operand error.
X'0C'	No terminal available.
X'10'	Receiving area is too short; only part of the information was supplied if the ALL, MONCS or BASIC operand was specified. In all other cases, nothing is supplied.
X'14'	Not all the information required is available.

Example 1

This example requests the logical terminal type and the type of screen overflow control. The receiving fields defined via the **TSTAT** macro (LOG and UEL) contain this information in hexadecimal form. The **DCSTA** macro is required for evaluating this information. **DCSTA** C,... is used to generate memory areas with symbolic addresses; these areas can be specified as receiving fields when the **TSTAT** macro is called.
All the symbolic names generated by the macro and their meanings are listed in the **DCSTA** macro description on page 381. For example, it is possible to read the type of overflow control from the `<PREFIX>OFLOW` field by evaluating the series of bits contained in it.

```
TSTAT1   START
         PRINT NOGEN
         BALR  3,0
         USING *,3
         TSTAT VDTYP,LOG,8 ------------------------------------------ (1)
         TSTAT OFLOW,UEL,8 ------------------------------------------ (2)
DTH1     TERM
LOG      DS    CL8
UEL      DS    CL8
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,tstat1), -
//         compiler-action=module-generation(module-format=llm), -
//         module-library=macexmp.lib, -
//         listing=parameters(output=*library-element(macexmp.lib,tstat1)), -
//         test-support=*aid
%  ASS6011 ASSEMBLY TIME: 245 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 78 MSEC
//end
%  ASS6012 END OF ASSEMBH
/load-executable-program library=macexmp.lib,element-or-symbol=tstat1, -
/     test-options=*aid
%  BLS0523 ELEMENT 'TSTAT1', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'TSTAT1', VERSION ' ' OF '<date> <time>' LOADED
/%in dth1<%d log %x, uel %x>
/%r
*** TID: 005000D8 *** TSN: 2QSE *********************************************
**
```

```
CURRENT PC: 00000022    CSECT: TSTAT1  **************************************
**
V'0000003E' = LOG      + #'00000000'  —————————————————————————————————  (3)
0000003E (00000000) 5B010000 00000000                     $.......
V'00000046' = UEL      + #'00000000'  —————————————————————————————————  (4)
00000046 (00000000) 02060000 00000000                     ........
```

(1) Logical type of terminal is requested.

(2) Type of overflow control is requested.

(3) The task runs on a data display terminal (bit 2^6 is set) at which the line, format and physical modes are all permitted (bits 2^0, 2^1 and 2^3 are set).

(4) The system (bit 2^5 =0) controls the overflow. When the screen is full, the system requests an acknowledgment at the terminal before overwriting the screen (bit 2^1 =1).

Example 2

```
  TSTAT2   START
           PRINT NOGEN
           BALR  3,0
           USING *,3
           TSTAT TCHAR,STATCHAR,8 ─────────────────────────────────────   (1)
           TSTAT PHDIM,PHYSAREA,8 ─────────────────────────────────────   (2)
           TSTAT LIDIM,FILIDIM,8 ──────────────────────────────────────   (3)
  DTH1     TERM
           PRINT GEN
           DCSTA C,TYPE=TCHAR ─────────────────────────────────────────   (4)
1 STATCHAR DS    0XL8
1 *
1 *              DEFINE TERMINAL CHARACTERISTICS FIELDS
1 *
1 STASTTCH DS    0XL8                     TERMINAL CHARACTERISTICS AREA
1 STAMNTCH DS    0XL8                     MINIMUM TERMINAL CHARICS. AREA
1 *
1 STAPTTYP DC    AL1(0)                   PARTNERTYPE
1 STADVTYP DC    AL1(0)                   DEVICE TYPE
1 STATCHR2 DC    AL1(0)                   TERMINAL CHARACTERISTICS BYTE 2
1 STATCHR3 DC    AL1(0)                   TERMINAL CHARACTERISTICS BYTE 3
1 STATCHR4 DC    AL1(0)                   TERM. CHARACTERISTIC BYTE 4 901
1 STATCHRS DC    AL1(0)                   TERM. CHAR FROM STATION     920
1 STACTRLU DC    AL1(0)                   CONTROL UNIT FOR PRINTER    701
1 STACHCAD DC    AL1(0)                   CENTRAL HARDCOPY ADDRESS
1 *
1 *              DEFINE PARTNER TYPES (PTTYP)
1 *
1 STADCAMP EQU   X'00'                  PARTNER IS A PROGRAM
1 STADCAMT EQU   X'01'                  PARTNER IS A TERMINAL
1 *
1          DCDEVCH STA
2 *
2 *              DEFINE DEVICE TYPES (DVTYP)
2 *
2 STAD8103 EQU   X'02'                 TELETYPE 8103
2 STAD8150 EQU   X'04'                 VIDEO TERMINAL 8150
2 STAD8153 EQU   X'05'     *NO VTSU*   VIDEO TERMINAL 8153
2 STADHOST EQU   X'08'                 INTELLIGENT PARTNER
2 STAD8151 EQU   X'15'                 VIDEO TERMINAL 8151
2 STAD8152 EQU   X'16'                 VIDEO TERMINAL 8152
2 STAD8110 EQU   X'17'                 SS-
8110                            00530000
2 STAD6154 EQU   X'18'     *NO VTSU*   VIDEO 8161 54 CHAR PER LINE
2 STAD6164 EQU   X'19'     *NO VTSU*   VIDEO 8161 64 CHAR PER LINE
2 STAD6180 EQU   X'1A'     *NO VTSU*   VIDEO 8161 80 CHAR PER LINE
```

```
2 STAD8161 EQU   X'1A'     *NO VTSU*   VIDEO 8161
2 STAD8121 EQU   X'1C'                 PRINTER STATION 8121
2 STADPT80 EQU   X'1D'     *AS 8103*   TELETYPE PT80
2 STAD1000 EQU   X'1E'     *AS 8103*   TELETYPE T1000
2 STADT100 EQU   X'23'     *AS 8103*   TELETYPE T100
2 STAD100E EQU   X'26'     *AS 8103*   FS100-E
2 STAD8122 EQU   X'2B'                 PRINTER STATION 8122
2 STAD8162 EQU   X'2C'                 VIDEO 8162
2 STAD8160 EQU   X'2D'                 VIDEO 8160
2 STAD8124 EQU   X'2E'                 PRINTER STATION 8124
2 STAD8167 EQU   X'2F'     *AS 8160*   VIDEO 8167
2 STADAP   EQU   X'30'     *AS HOST*   AP-STATION
2 STAD9750 EQU   X'35'                 VIDEO 9750 OR 9749
2 STAD9003 EQU   X'36'                 PRINTER STATION 9003
2 STAD9770 EQU   X'39'     *AS 8151*   DS 9770
2 STAD9002 EQU   X'3B'                 PRINTER STATION 9002
2 STAD3974 EQU   X'3D'                 VIDEO TERMINAL 3974
2 STAD9751 EQU   X'3F'     *AS 8160*   DSS 9751
2 STAD9752 EQU   X'40'     *AS 9750*   DSS 9752
2 STAD9753 EQU   X'41'     *AS 9750*   DSS 9753
2 STAD9001 EQU   X'42'                 PRINTER 9001
2 STAD9731 EQU   X'43'     *AS 3974*   GRAFIC STATION 9731
2 STAD9004 EQU   X'45'                 PRINTER 9004
2 STAD9754 EQU   X'4C'     *AS 8160*   VIDEO 9754
2 STAD9755 EQU   X'4E'                 DSS-9755
2 STAD9763 EQU   X'4F'                 DSS-9763
2 STADBTXF EQU   X'55'     *AS HOST*   BTX-STATION T-3000 (FELDVERS.)
2 STADBTXE EQU   X'56'     *AS HOST*   BTX-EDITIER-STATION (DIENST)
2 STADBTXA EQU   X'57'     *AS HOST*   BTX-ABFRAGE-STATION (DIENST)
2 STADUTC  EQU   X'5A'                 UTC FUER TELETEX
2 STAD9012 EQU   X'5B'                 PRINTER 9012
2 STAD9013 EQU   X'5C'                 PRINTER 9013
2 STAD3270 EQU   X'5E'                 DSS-3270
2 STAD0131 EQU   X'65'                 PRINTER 9001-31
2 STAD0189 EQU   X'66'                 PRINTER 9001-8931
2 STAD9022 EQU   X'68'                 PRINTER 9022
2 STAD1118 EQU   X'6B'                 PRINTER 9011-18
2 STAD1119 EQU   X'6C'                 PRINTER 9011-19
2 STAD3287 EQU   X'6E'                 PRINTER 3287
2 STADPCL  EQU   X'70'                 PRINTERS PCL
2 STAD9021 EQU   X'70'                 PRINTERS 9021 / 9022-200, HP LJ
2 STAD9014 EQU   X'72'                 PRINTER 9014
2 STAD9026 EQU   X'73'                 PRINTER 9026 (HDLC,COMP.9025)
2 STADTNO8 EQU   X'74'                 Telnet without overflow 8-bit
2 STADTOV8 EQU   X'75'                 Telnet with overflow 8-bit
2 STADFE   EQU   X'78'                 FRONT-END TERMINAL (FHS-DOORS)
2 *
2 *              DEFINE TERMINAL CHARACTERISTICS BYTE 2 (TCHR2) BITS
```

```
2 *
2 STATC2EX EQU   8                         SECONDARY CHARACTER SET
2 STATC2LC EQU   32                        LOWER CASE
2 STATC2DT EQU   64                        GERM KEYB WITH GERM NAT CHAR
2 STATC2DF EQU   128                       BYTE 2 DEFINED
2 *
2 *              DEFINE TERMINAL CHARACTERISTICS BYTE 3 (TCHR3) BITS
2 *
2 STATC3H1 EQU   1                         HARDCOPY BIT 1  (LOCAL)
2 STATC3H2 EQU   2                         HARDCOPY BIT 2  (CENTRAL)
2 STATC3HC EQU   3                         HARDCOPY BITS
2 STATC3IC EQU   4                         IDENTITY CARD READER
2 STATC3FD EQU   8                         FLOPPY DISK
2 STATC3AP EQU   16                        APL CAPABILITY
2 STATC3GF EQU   32                        GRAPHICS
2 STATC3DZ EQU   64                        DEZENTRAL FORMATING
2 STATC3DF EQU   128                       BYTE 3 DEFINED
2 *
2 *              DEFINE TERMINAL CHARACTERISTICS BYTE 4 (TCHR4) BITS
2 *
2 STATC4CO EQU   1                         4 COLOURS(ITALIC/HALFBRIGHT)
2 STATC4ZF EQU   2                         NEW ZAT AND FAT POSSIBLE
2 STATC4ST EQU   4                         STATUS QUERY POSSIBLE
2 STATC4HI EQU   8                         HARDWARE INFOLINE AVAILABLE
2 STATC4C8 EQU   16                        8 COLOURS
2 STATC4HP EQU   32                        HP LASER JET II
2 STATC4DF EQU   128                       BYTE 4 DEFINED
2 *
2 *              DEFINE TERM CHAR FROM STATION   BYTE   (TCHRS) BITS
2 *
2 STATCSDT EQU   1                         GERMAN KEYBOARD
2 STATCSHC EQU   2                         LOCAL HARDCOPY PRINTER
2 STATCSIC EQU   4                         ID-CARD READER
2 STATCDOR EQU   8                         DOORS capability (reserved)
2 STATCDSK EQU   16                        DESK capability
2 STATCECC EQU   32                        ENCRYPTION capability (res)
2 STATCPER EQU   64                        Permanent ENCRYPTION reques
2 STATCSDF EQU   128                       TERM CHAR FROM STAT RECEIVED
2                *,DCDEVCH    200    960821
1 *
1                *,DCSTA      201    970513
  PHYSAREA DCSTA C,TYPE=PHDIM ——————————————————————————————————————————  (5)
1 PHYSAREA DS    0XL8
1 *
1 *              DEFINE PHYSICAL TERMINAL ATTRIBUTES FIELDS
1 *
1 STASTPV  DS    0XL8                      PHYSICAL TERMINAL ATTR. AREA
1 STAMNPV  DS    0XL8                      MINIMUM PHYS. TERM. ATTR. AREA
```

```
1 *
1 STALLEN  DC    H'0'                        PHYSICAL LINE LENGTH
1 STANOLIN DC    H'0'                        PHYSICAL NUMBER OF LINES
1 STAMAXDB DC    H'0'                        MAX. PHYSICAL DEVICE BUFFER
1          DC    2AL1(0)                     RESERVED FOR FUTURE DEVELOPMENT
1                *,DCSTA      201    970513
           DCSTA C,FI,TYPE=LIDIM ——————————————————————————————————————  (6)
1 FILIDIM  DS    0XL8
1 *
1 *              DEFINE VIRTUAL TERMINAL ATTRIBUTES FIELDS
1 *
1 FISTLV   DS    0XL8                        VIRTUAL TERMINAL ATTR. AREA
1 FIMNLV   DS    0XL8                        MINIMUM VIRTUAL TERM ATTR AREA
1 *
1 FILLLEN  DC    H'0'                        VIRTUAL LINE LENGTH
1 FILNOLN  DC    H'0'                        VIRTUAL NUMBER OF LINES
1 FILMAXB  DC    H'0'                        MAXIMUM VIRTUAL DEVICE BUFFER
1          DC    2AL1(0)                     RESERVED FOR FUTURE DEVELOPMENT
1                *,DCSTA      201    970513
           END
```

(1) Physical type of terminal is requested. The destination area which is generated by means of the **DCSTA C,...** macro is called STATCHAR by default.

(2) Physical dimensions of terminal are requested. PHYSAREA is the name of the destination area selected by the user.

(3) Logical dimensions of terminal are requested. The default prefix “STA” is replaced by the prefix “FI” in the name of the FILIDIM destination area.

(4) The destination area for information on the physical type is generated together with the symbolic constants for checking the bit values.

(5) The destination area for information on the physical dimensions is generated.

(6) The destination area for information on the logical dimensions is generated. “FI” is to be the prefix for the field names (default: “STA”).

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,tstat2), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,tstat2)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 389 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 112 MSEC
//end
%  ASS6012 END OF ASSEMBH
/load-executable-program library=macexmp.lib,element-or-symbol=tstat2, -
/     test-options=*aid
%  BLS0523 ELEMENT 'TSTAT2', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'TSTAT2', VERSION ' ' OF '<date> <time>' LOADED
/%in dth1<%d statchar %xl8, physarea %xl8, filidim %xl8>
/%r
*** TID: 005000D8 *** TSN: 2QSE **********************************************
**
CURRENT PC: 00000036    CSECT: TSTAT2  **************************************
**
V'00000052' = STATCHAR + #'00000000'
00000052 (00000000) 0135A081 88000000                         ...ah...  ———  (7)
V'0000005A' = PHYSAREA + #'00000000'
0000005A (00000000) 00500018 17FF0000                         .&...~..   ——  (8)
V'00000062' = FILIDIM  + #'00000000'
00000062 (00000000) 00500018 07800000                         .&......   ——  (9)
```

(7) The process is running on a 9755 Data Display Terminal.

(8) X'0050' The physical line length is 80 characters.
X'0018' The physical number of lines is 24 lines.
X'17FF' The physical device buffer has a capacity of 6143 characters.

(9) X'0050' The logical line length is 80 characters (LINE mode).
X'0018' The logical number of lines is 24 lines (LINE mode).
X'0780' The logical character buffer comprises 1920 characters
(= 24 lines x 80 columns).

TYPIO – Write message to console

General

Application areas:	Data terminal communication; see page 160 Message system; see page 161 Communication; see page 163
Macro type:	Type S, MF format **1**: standard/E/L form; see page 29

For a detailed description of physical and logical consoles and their use in BS2000 refer to the “Introduction to System Administration” [10].

Macro description

The **TYPIO** macro is used to output a message on the console and to receive a message from the console.

Macro format and description of operands

```
TYPIO
--------------------------------------------------------------

       ⎧ addr1                      ⎫
MSG=   ⎨ (r1)                       ⎬
       ⎩ (base1,[index1],[disp1])   ⎭

                   ⎧ addr2                      ⎫
[,REPLY=(length,   ⎨ (r2)                       ⎬  )]
                   ⎩ (base2,[index2],[disp2])   ⎭
,SHORT=NO / YES

             ⎧ 'destcode'                 ⎫
[,UCDEST=    ⎨ addr3                      ⎬  ]
             ⎪ (r3)                       ⎪
             ⎩ (base3,[index3].[disp3])   ⎭

                   ⎧ addr4 ⎫
,MF=S / L / (E,    ⎨ (1)   ⎬  )
                   ⎩ (r4)  ⎭
```

MSG=

Specifies the address of an output area for the output message (variable-length record). Message length ≤ 230 bytes; messages of a length between 231 and 251 bytes are truncated; messages even longer than that are not transferred (length error).

addr1

Symbolic address (name) of the output area.

(r1)

Register containing the address value “addr1”.

(...)

Indirect adress for addr1.
base1 = base register 1; index1 = index register 1; disp1 = displacement 1

REPLY=

Specifies that a reply is expected. This operand defines the address of the input area for the reply (variable-length record).
Reply length ≤ 72 bytes.

length

Length of input area (anticipated reply length + 4); 4 ≤ length ≤ 76.

- Longer replies are truncated.
- Shorter replies are entered left-justified and a byte with value X'00' is appended.
- length < 4 and length > 251 both result in a length error (RC = X'0C').
- 77 ≤ length ≤ 251 results in a length error (RC = X'04') and is treated as length = 76.

=addr2

Symbolic address (name) of the input area.

=(r2)

Register containing the address value “addr2”.

(...)

Indirect adress for addr2.
base2 = base register 2; index2 = index register 2; disp2 = displacement 2

SHORT=

Specifies whether the message is to be output with an abbreviated or unabbreviated header (only effective for output to physical consoles).

NO

Default setting: specifies that the message is output in unabbreviated form.

YES

The message header is output in the form `%xxxx_` where:
% = message without reply
xxxx = source code/name/TSN and

SHORT=YES is permitted only if REPLY has not been specified.

UCDEST=
UCON output destination. The destination of the message may be specified as follows:

- mnemonic device name for a specific console
- routing code for consoles and user tasks to which a specific job area has been assigned
- authorization name for an authorized user task.

'destcode'
The following specifications are possible (Entries must be enclosed in single quotes):

- destcode = (mn)
  where mn = 2-character mnemonic device name
- destcode = <x
  where x = routing code (be sure to specify the < character)
- destcode = name of the user task (4 characters)

addr3
Symbolic address (name) of a field (word) containing the entry for “destcode”.

(r3)
Register containing the entry for “destcode”.

(...)
Indirect adress for addr3.
base3 = base register 3; index3 = index register 3; disp3 = displacement 3

Note

All entries must be entered left-justified.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

In the E form of the macro, the address of the parameter list is written to register R1 during macro execution.

Notes on the macro call

- When using indirect addressing (operands MSG, REPLY, UCDEST) the commas must always be entered.
- Format of the output or input area (operand MSG or REPLY):

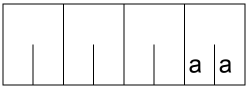
  Byte 0-1: Record length field (length of the message/reply + 4); when REPLY is specified, this field is supplied with values by **TYPIO**.
  Byte 2-3: reserved
  Byte 4-n: Text of message/reply

```
Example for an output area:          For an input area:
message    DC   Y(mend-message)      reply       DS    0CL54
           DS   CL2                  lenfield    DS    CL2
           DC   C'message-text'                  DS    CL2
mend       EQU  *                    rtext       DS    CL50
```

- The macro can be used in reentrant programs, provided that only register entries are used and the L form and E form are called individually.
- During execution of the **TYPIO** macro, the contents of registers R0 and R1 are overwritten. The contents of register R0 are overwritten with binary zeros. The start address of the parameter list is written to register R1 (in the E form of the macro). For this reason, neither of these registers should be used for storing other values.

Return information and error flags

R15:



A return code relating to the execution of the TYPIO macro (format 2) is transferred in the rightmost byte of register R15. The remaining bytes are deleted.

X'aa'	Meaning
X'00'	Normal execution.
X'04'	TYPIO completed. Message or reply length truncated Message: length between 230 and 251specified Reply: length between 77 and 251 specified
X'08'	Address error: address was given via register R0 or R1, or an address (addr1 - addr3) is at least partially not in the user area.
X'0C'	Length error: message length = 0 or length greater than 251 bytes or negative lengths specified.
X'10'	Message output not possible (can occur, for example, during generation or termination of the calling task).

UNBIND – Unload and unlink objects

General

Application area:	Linking and loading; see page 47
Macro type:	Type S, MF format **2**: standard/C/D/L/E/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **UNBIND** macro is used during the program run to release memory occupied by objects that are no longer needed. The object can be a context, a load unit, an LLM or an object module (OM). The symbols in the unloaded objects are then no longer available. The occupied memory space is released pagewise only (in units of 4Kb). The space is returned to memory management only if it is not required by any other modules on the same page. Otherwise DBL notes the free areas and uses them at the next opportunity.

Control sections (CSECTs) and entry points (ENTRYs) in the object can optionally be unlinked, i.e. external references to these symbols are treated as unresolved external references by DBL. An object can be loaded only within a context and then only if LDINFO=REF (**BIND** macro) or LOAD-INFORMATION=*REFERENCES (LOAD-PROGRAM command) was specified when the object was loaded.

Macro format and description of operands

```
UNBIND

[⎧ CONTEXT=name        ⎫]
 ⎩ CONTXT@=addr / (r)  ⎭

[,⎧ UNIT=name          ⎫]
  ⎩ UNIT@=addr / (r)   ⎭

[,⎧ MODULE=name        ⎫]
  ⎩ MODULE@=addr / (r) ⎭

[,⎧ PGMVERS=*STD / version ⎫]
  ⎩ PGMVER@=addr / (r)     ⎭

,UNLINK=NO / YES

,MSG=*DBLOPT / INFORMATION / WARNING / ERROR / NONE

,MF=S / C / D / E / L / M

[,PARAM=addr / (r)]

,PREFIX=P / p

[,LABEL=name]
```

CONTEXT=name
Specifies the name of the context to be unloaded or in which objects are to be unloaded. Objects can be:

- a load unit specified by UNIT or
- a module specified by MODULE.

“name” can be up to 32 characters long and must not begin with “$” or “#”.
If the operand is not used the name “LOCAL#DEFAULT” is assumed by default.

CONTXT@=
May be specified only if MF=M.
Specifies the address of a field which contains the name of the context to be unloaded.

addr
Address of an field which contains the name.

(r)
Register containing the address value “addr”.

UNIT=name

Specifies the name of the load unit to be unloaded or in which a module specified by MODULE is to be unloaded. “name” is the name of the load unit which was specified in the **BIND** macro at the time of loading. This can be:

- the name defined with the UNIT@ or UNIT operand or
- the name defined with the SYMBOL@ or SYMBOL operand if UNIT@ or UNIT was not specified.

“name” can be up to 32 characters long.

UNIT@=

May be specified only if MF=M.
Specifies the address of a field which contains the name of the load unit to be unloaded.

addr
Address of a field which contains the name.

(r)
Register containing the address value “addr”.

MODULE=name

Specifies the name of the module to be unloaded. The module can be an LLM or an OM. If an LLM, its internal name must be specified. “name” can be up to 32 characters long.

MODULE@=

May be specified only if MF=M.
Specifies the address of a field containing the name of the module which is to be unloaded.

addr
Address of a field which contains the name.

(r)
Register containing the address value “addr”.

UNLINK=

Specifies whether or not control sections (CSECTs) and entry points (ENTRYs) are to be unlinked in the unloaded object. External references to unlinked symbols are then treated as unresolved external references by DBL. Unlinking is only possible if the LDINFO=REF operand was specified in the **BIND** macro at the time of loading.
Symbols in a context which refer to another context cannot be unlinked. Unlinking is only possible within the same context.

NO
Default setting: symbols will not be unlinked.

YES
Symbols will be unlinked.

PGMVERS=
Specifies the program version to be unloaded.

<u>*STD</u>
Default setting: signifies that no version specification is to be taken into account during unloading.

version
The version specification may be up to 24 characters long.
If the DBL does not find this program version, the program specified is not unloaded.

PGMVER@=
May be specified only if MF=M.
Specifies the address of a field containing the program version.

addr
Address of a field which contains the name.

(r)
Register containing the address value “addr”.

MSG=
Specifies the lowest message class; messages at and above this level will be output.

<u>*DBLOPT</u>
Default setting: the parameter value is taken from the last call of the MODIFY-DBL-PARAMETERS command. If a value for the parameter has not yet been set using the MODIFY-DBL-PARAMETERS command, MSG=INFORMATION applies.

INFORMATION
All classes of message will be output.

WARNING
Only messages of the WARNING and ERROR classes will be output. Messages of the INFORMATION message class will not be output.

ERROR
Only messages of the ERROR class will be output.

NONE
No messages will be output.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM and PREFIX, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro
(see section “S-type macros” on page 29).

LABEL=name
May be specified only if MF=M
Name of the structure, i.e. the DSECT which describes the operand list of the **UNBIND** macro. The operand is mandatory if there is no valid USING statement for the definition of the base address register for the DSECT of the parameter list. The LABEL operand must be specified in conjunction with the PARAM operand. Both operands are used to produce a valid USING statement.

The following may be specified for “name”:

1. The name specified in the name field of a preceding macro `name UNBIND MF=D`
2. The name “xPBUNDS” if no “name” has already been specified, where “x” is the value of the PREFIX operand of a preceding macro `PBUNBIND MF=D, PREFIX=x`
   The default value for “x” is “P”.
3. The name of the longer DSECT containing the parameter list of the **UNBIND** macro if the macro `UNBIND MF=C` was specified earlier.

Notes on the macro call

– The CONTEXT, UNIT and MODULE operands can be specified together in the same macro in order to expedite the search by DBL or to resolve name conflicts in cases where load units or modules have the same name. By default, DBL always chooses the first name it finds.

– The delayed resolution of external references (DELAY) and the unlinking (UNLINK) are restricted to one context. References which have been resolved by symbols in more than one context cannot be unlinked.

– **UNBIND** can be used to unload only objects which were loaded with the **BIND** macro or with the LOAD- or START-EXECUTABLE-PROGRAM (or LOAD- and START-PROGRAM) command. Shared code which was loaded into a common memory pool with the **ASHARE** macro cannot be unloaded with **UNBIND**; the **DSHARE** macro must be used for this.

– A module belonging to a list name unit cannot be unloaded independently of the list name unit.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the UNBIND macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally.
X'0C'	X'01'	X'0018'	A reserved field in the parameter list is not preset with zeros.
X'0C'	X'01'	X'0100'	Illegal parameter combination in the parameter list. Blanks must be entered for the values of CONTEXT, UNIT and MODULE.
X'0C'	X'01'	X'015C'	The specified context is not present.
X'0C'	X'01'	X'0170'	The specified load unit is not present.
X'0C'	X'01'	X'0174'	The specified module is not present.
X'0C'	X'01'	X'0178'	A module belonging to a list name unit cannot be unloaded independently of the list name unit.
X'0C'	X'01'	X'0198'	Illegal context name. The first character is not a letter.
X'0C'	X'01'	X'0204'	Inconsistencies in the DBL memory management tables (system error).
X'0C'	X'01'	X'0208'	Inconsistencies in the DBL tables (system error)
X'0C'	X'01'	X'0300'	Error during RETMEM processing (system error).
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported.
X'00'	X'03'	X'FFFF'	The interface version is not supported.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Example

During the UNBIND1 program run, a **BIND** macro is issued to load a second control section, BIND3, as an overlay. BIND3 is stored as an object module in library MACEXMP.LIB. Both control sections are to execute in 31-bit addressing mode. UNBIND1 is to be loaded below, and BIND3 above, the 16-MB boundary. After the **BIND** macro is called BIND3 is to execute first. Following the execution of BIND3 a return branch is to be made to UNBIND1 and module BIND3 is to be unloaded by means of the **UNBIND** macro.

```
UNBIND1  START
UNBIND1  AMODE 31 ──────────────────────────────────────────────────────   (1)
UNBIND1  RMODE 24
         BALR  3,0
         USING *,3
         USING BINDDS,6 ─────────────────────────────────────────────────   (2)
         USING UNBDS,7 ──────────────────────────────────────────────────   (3)
         ST    3,AREA11
         UNPK  AREAH,AREA1
         MVC   AREAA(8),AREAH
WROUT1   WROUT OUT,ERROR,PARMOD=31 ─────────────────────────────────────   (4)
BACK     LA    12,UNBIND
BIND     BIND  MF=E,PARAM=BINDPAR ──────────────────────────────────────   (5)
         LA    6,BINDPAR
         CLC   XBINRET,=X'00000000' ────────────────────────────────────   (6)
         BE    UNBIND
         MVC   OUT+5(28),='BIND ERROR!                 '
         WROUT OUT,ERROR,PARMOD=31 ─────────────────────────────────────   (7)
         B     ERROR
UNBIND   UNBIND MF=E,PARAM=UNBPAR ──────────────────────────────────────   (8)
         LA    7,UNBPAR
         CLC   YUNBRET,=X'00000000' ────────────────────────────────────   (9)
         BE    MVC
         MVC   OUT+5(28),='UNBIND ERROR!               '
         WROUT OUT,ERROR,PARMOD=31 ─────────────────────────────────────  (10)
         B     ERROR
MVC      MVC   OUT+5(28),='UNBIND PROCESSED            '
         WROUT OUT,ERROR,PARMOD=31 ─────────────────────────────────────  (11)
         MVC   OUT+5(28),='RETURN TO UNBIND1           '
         WROUT OUT,ERROR,PARMOD=31
ERROR    TERM
*****************
OUT      DC    Y(OUTE-OUT)
         DS    CL3
         DC    C'UNBIND1: BASE REG.= '
AREAA    DS    CL8
OUTE     EQU   *
AREA     DS    0F
AREA1    DS    0CL5
```

```
AREA11   DS    CL4
AREA12   DC    C'0'
         DS    0F
AREAH    DS    CL9
BINDPAR BIND MF=L,SYMBOL=BIND3,SYMBLAD=BIND3@,BRANCH=YES,PROGMOD=ANY,* ─ (5)
               LIBLINK=PLAMLIB
UNBPAR   UNBIND MF=L,MODULE=BIND3 ────────────────────────────────────── (8)
BIND3@   DS    A
BINDDS   BIND  MF=D,PREFIX=X ────────────────────────────────────────── (12)
UNBDS    UNBIND MF=D,PREFIX=Y ───────────────────────────────────────── (13)
         END


BIND3    CSECT ──────────────────────────────────────────────────────── (14)
         PRINT NOGEN
BIND3    AMODE ANY ──────────────────────────────────────────────────── (15)
BIND3    RMODE ANY
         BALR  4,0
         USING *,4
         ST    4,AREA11
         UNPK  AREAH,AREA1
         MVC   AREAA(8),AREAH
         WROUT OUT,ERROR,PARMOD=31 ──────────────────────────────────── (16)
         BR    12
ERROR   TERM
*****************
OUT      DC    Y(OUTE-OUT)
         DS    CL3
         DC    C'BIND3:   BASE REG.= '
AREAA    DS    CL8
OUTE     EQU   *
AREA     DS    0F
AREA1    DS    0CL5
AREA11   DS    CL4
AREA12   DC    C'0'
AREAH    DS    CL9
         END
```

(1) The attribute AMODE=31 is defined for control section UNBIND1. The attribute RMODE=24 means that UNBIND1 will always be loaded below the 16-MB boundary.

(2) Register 6 is assigned to the assembler as the base address register for addressing the DSECT for the operand list of the **BIND** macro, which is generated at the symbolic address BINDDS as a result of a **BIND** macro specifying MF=D.

(3) Register 7 is assigned to the assembler as the base address register for addressing the DSECT for the operand list of the **UNBIND** macro, which is generated at the symbolic address UNBDS as a result of an **UNBIND** macro specifying MF=D.

(4) The contents of the base register for UNBIND1 are output to indicate the addressing mode and the load address.

(5) The **BIND** macro is called in its E form at the symbolic address BIND. At this point in the program, therefore, only the instruction code is generated. The associated operand list is created at the symbolic address BINDPAR by means of a **BIND** macro specifying MF=L. As a result of the operand values specified in the list, the **BIND** macro causes the following to happen at program runtime:

- the CSECT BIND3 (SYMBOL=BIND3) is reloaded from the library assigned with the link name PLAMLIB (LIBLINK=PLAMLIB)
- the start address of BIND3 is stored in field BIND3@ (SYMBLAD=BIND3@)
- the 31-bit addressing mode is set for BIND3 (PROGMOD=ANY)
- the program run is continued in BIND3 after BIND3 has been loaded (BRANCH=YES).

(6) Following execution of the **BIND** macro, a check is made to verify that the XBINRET field of the standard header contains the return code X’00000000’, which indicates error-free execution of the macro. The name XBINRET originates from the DSECT that was generated under the symbolic address BINDDS as a result of a **BIND** macro specifying MF=D and PREFIX=X (see point 12 below). This DSECT describes the layout of the operand list of the **BIND** macro. The symbolic names of the DSECT can be used for addressing within the operand list once the assigned base address register (in this case, register 6) has been loaded with the start address of the operand list (in this case, BINDPAR).

(7) If the **BIND** macro does not execute without error, an error message is written to SYSOUT and the UNBIND1 program run is terminated.

(8) The **UNBIND** macro is called in its E form at the symbolic address UNBIND. At this point in the program, therefore, only the instruction code is generated. The associated operand list is created at the symbolic address UNBPAR by means of an **UNBIND** macro specifying MF=L. As a result of the operand MODULE=BIND3 specified in the list, the **UNBIND** macro causes the BIND3 module to be unloaded.

(9) Following execution of the **UNBIND** macro, a check is made to verify that the YUNBRET field of the standard header contains the return code X’00000000’, which indicates error-free execution of the macro. The name YUNBRET originates from the DSECT that was generated under the symbolic address UNBDS as a result of an **UNBIND** macro specifying MF=D and PREFIX=Y (see point 13 below). This DSECT describes the layout of the operand list of the **UNBIND** macro. The symbolic names of the DSECT can be used for addressing within the operand list once the assigned base address register (in this case, register 7) has been loaded with the start address of the operand list (in this case, UNBPAR).

(10) If the **UNBIND** macro does not execute without error, an error message is written to SYSOUT and the UNBIND1 program run is terminated.

(11) Messages written to SYSOUT indicate that program execution has continued in UNBIND1 and module BIND3 has been unloaded.

(12) The **BIND** macro specifying MF=D generates a DSECT which describes the layout of the operand list of the **BIND** macro. The operand PREFIX=X causes the letter X to be prefixed to all symbolic names in this DSECT (field names and equates).

(13) The **UNBIND** macro specifying MF=D generates a DSECT which describes the layout of the operand list of the **UNBIND** macro. The operand PREFIX=Y causes the letter Y to be prefixed to all symbolic names in this DSECT (field names and equates).

(14) The CSECT statement defines the control section BIND3.

(15) AMODE=ANY indicates to the operating system that BIND3 can execute in 24-bit or 31-bit addressing mode.

(16) The contents of the base register for BIND3 are output to indicate the addressing mode and the load address.

Runtime listing

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,unbind1), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,unbind1))
%  ASS6011 ASSEMBLY TIME: 585 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 200 MSEC
```

```
//compile source=*library-element(macexmp.lib,bind3), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,bind3))
%  ASS6011 ASSEMBLY TIME: 169 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 83 MSEC
//end
%  ASS6012 END OF ASSEMBH
/add-file-link link-name=plamlib,file-name=macexmp.lib ─────────────── (17)
/start-executable-program library=macexmp.lib,element-or-symbol=unbind1  (18)
%  BLS0523 ELEMENT 'UNBIND1', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'UNBIND1', VERSION ' ' OF '<date> <time>' LOADED
UNBIND1: BASE REG.= 80000002 ──────────────────────────────────────── (19)
BIND3  : BASE REG.= 81000002 ──────────────────────────────────────── (20)
UNBIND PROCESSED
RETURN TO UNBIND1 ─────────────────────────────────────────────────── (21)
```

(17) The file link name used in the BIND call (5) is assigned.

(18) DBL is invoked to link, load and start the program.

(19) The contents of the base register for UNBIND1 are output. 31-bit addressing is set (bit 2^{31} = 1); the load address is below the 16-MB boundary.

(20) DBL has loaded the CSECT BIND3. The contents of the base register for BIND3 are output. 31-bit addressing is set (bit 2^{31} = 1); the load address is above the 16-MB boundary.

(21) Following the return from BIND3, the program run is continued in UNBIND1 and module BIND3 is unloaded.

VMGINF – Output information on VM2000 operation

General

Application area: Requesting and accessing lists and tables; see page 155
Macro type: Type S, MF format **3**: D/C/E/L form; see page 29

The VM2000 virtual machine system allows different, completely separate system environments to run on *one* server, with a performance comparable to a “native” system. VM2000 thus increases the number of possible applications of a server and the degree to which it is utilized. For further information, see the “VM2000” manual [17].

Macro description

The **VMGINF** macro tells the caller whether BS2000 is running under VM2000 (<PREFIX><MACID>VIND field of the parameter list). If this is the case, the caller is provided with the following information on the VM2000 system, under which BS200 is currently running or on which the IPL of BS2000 has been executed:

- the index (VM-INDEX) and the name (VM-NAME) of the virtual machine
- whether the system is a monitor system
- the SYSID (VM-CONFIGURATION-ID) of the monitor system.
- specific VM privileges and states of the guest system
- the version of VM2000
- the BCAM name of the monitor system
- the version of the monitor system
- Xen domain ID (servers with x86 architecture)

Macro format and description of operands

VMGINF
SRVUNIT = *STD / *INITIAL / *CURRENT ,MF=D / C / E / L [,PARAM=addr / (r)] ,PREFIX=V / p ,MACID=MGI / macid

SRVUNIT=
Specifies the Server Unit whose data is to be output.

*STD
Currently valid default setting of the Server Unit.

*INITIAL
Server Unit on which IPL was performed for BS2000.

*CURRENT
Server Unit on which BS2000 is currently running.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form or D form of the macro and additionally a MACID in the C form (see section “S-type macros” on page 29).

ℹ After calling the macro, the user must first establish whether the system is running under VM2000. Only then can further output information be evaluated.

Layout of the data area

```
           VMGINF MF=D
1          MFTST MF=D,PREFIX=V,MACID=MGI,ALIGN=F,                        C
1                DMACID=MGI,SUPPORT=(E,D,C,M,L)
2 VMGI     DSECT ,
2                *,##### PREFIX=V, MACID=MGI #####
1 VMGIUNIT    EQU   137                    unit number
1 *
1 VMGIFC04    EQU   4                      function number
1 *
1 VMGIVR02    EQU   2                      version number
1 *
1 *   parameterarea description
1 VMGIPA   DS    0F                  begin of parameterarea    _INOUT
1 VMGIHDR  FHDR  MF=(C,VMGI),EQUATES=NO                    Standardheader
2 VMGIHDR  DS    0A
2 VMGIFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 VMGIIFID DS    0A              0   INTERFACE IDENTIFIER
2 VMGIFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 VMGIFCT  DS    AL1             2   FUNCTION NUMBER
2 VMGIFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 VMGIRET  DS    0A              4   GENERAL RETURN CODE
2 VMGISRET DS    0AL2            4   SUB RETURN CODE
2 VMGISR2  DS    AL1             4   SUB RETURN CODE 2
2 VMGISR1  DS    AL1             5   SUB RETURN CODE 1
2 VMGIMRET DS    0AL2            6   MAIN RETURN CODE
2 VMGIMR2  DS    AL1             6   MAIN RETURN CODE 2
2 VMGIMR1  DS    AL1             7   MAIN RETURN CODE 1
2 VMGIFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1 *   main return codes
1 VMGIMSCC    EQU   0                      function executed
1 VMGIMPAR    EQU   1                      parameter error
1 VMGIMINT    EQU   2                      internal error
1 VMGIMTIM    EQU   7                      timeout error
1 VMGIMANA    EQU   64                     VM2000 agent not available
1 VMGIMXNA    EQU   65                     xend not available
1 *
```

```
1 VMGIVIND    DS    FL1                  VM2000 indicator
1 *   VM2000 indicator set
1 VMGIVM2R    EQU   232                  VM2000 running
1 VMGIVM2N    EQU   213                  VM2000 not running
1 *
1 VMGISIND    DS    AL1                  status indicator
1 VMGIV2MO    EQU   X'80'                monitor system
1 VMGILTSY    EQU   X'40'                local time syn via adjust
1 *                                      time
1 VMGIMPAD    EQU   X'20'                MP grade adjustment possible
1 VMGIPIOM    EQU   X'10'                VM-PRIV: DYNAMIC-IOREC
1 VMGIPGIO    EQU   X'08'                VM-PRIV: VMGLOB-IOREC
1 VMGIUNU0    EQU   X'04'                unused
1 VMGIGS2N    EQU   X'02'                GS-UNIT2 not in use
1 VMGIGS1N    EQU   X'01'                GS-UNIT1 not in use
1 VMGIVCID    DS    X                    VM configuration ID
1 *
1 VMGIVM_IDENT DS    0XL9                 VM identification
1 VMGIVMIX    DS    X                    VM index
1 VMGIVMNM    DS    CL8                  VM name
1 *
1 VMGIV1IN    DS    AL1                  valid indicator 1
1 VMGIV1O1    EQU   X'80'                :S: _OUT_11 valid
1 VMGIV1O2    EQU   X'40'                :S: _OUT_12 valid
1 VMGIV1O3    EQU   X'20'                :S: _OUT_13 valid
1 VMGIV1O4    EQU   X'10'                :S: _OUT_14 valid
1 VMGIUNU1    EQU   X'0F'                unused
1 VMGIV2IN    DS    AL1                  valid indicator 2
1 VMGIUNU3    EQU   X'FF'                unused
1 VMGIVVRS    DS    CL6                  VM2000 version (Vxx.xx)
1 *                                      _OUT_11
1 VMGISIN2    DS    AL1                  status indicator 2
1 VMGIPIDA    EQU   X'80'                VM-PRIV: IMPL-DEV-ASSIGN
1 *                                      _OUT_12
1 VMGIPIOP    EQU   X'40'                VM-PRIV: IO-PRIORITY _OUT_12
1 VMGIPIOR    EQU   X'20'                VM-PRIV: IO-RESET _OUT_12
1 VMGICHMF    EQU   X'10'                CHN-MON-FCL ACTIVE FOR VM
1 *                                      _OUT_12
1 VMGIMBCA    EQU   X'08'                BCAM ACTIVE IN MONITOR
1 *                                      _OUT_12
1 VMGIPASA    EQU   X'04'                VM-PRIV: AUTO-SNAP-ASSIGN
1 *                                      _OUT_12
1 VMGIRSCS    EQU   X'02'                RSC SUPPORTED _OUT_14
1 VMGIUNU2    EQU   X'01'                unused
1 VMGIMBCN    DS    CL8                  monitor BCAM name _OUT_12
1 VMGIMOVS    DS    CL10                 monitor OSD version _OUT_12
1 VMGIDOID    DS    XL8                  Xen domid _OUT_13
1 VMGIRES1    DS    XL43                 reserverd
```

```
1 VMGISUI     DS    FL1                  server unit indicator
1 *   Server unit indicator set
1 VMGISUIS    EQU   0                    standard
1 VMGISUII    EQU   1                    initial
1 VMGISUIC    EQU   2                    current
1 *
1 VMGICID     DS    FL1                  caller identifier (internal
1 *                                      use)
1 *   Caller identifier set
1 VMGICIDS    EQU   0                    system
1 VMGICIDU    EQU   1                    user
1 *
1 VMGI#       EQU   *-VMGIHDR
```

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the VMGINF macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'01'	X'0001'	Parameter error
X'00'	X'20'	X'0002'	Internal error
X'0C'	X'40'	X'0007'	Timeout error
X'0C'	X'82'	X'0040'	VM2000 agent not available
X'0C'	X'82'	X'0041'	XEND not available

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

The names of the output fields are given in the parameter list.

VPASS – Variable-length pass

General

Application area: Starting, interrupting and terminating; see page 72
Macro type: Type R; see page 28

Macro description

The **VPASS** macro is used to relinquish control of the processor for a specified period, i.e. the user task is set to a wait state. The waiting period specification can be made relative to the CPU rate.

Macro format and description of operands

VPASS
number / (1) ,MSEC=N / Y ,CPUDEP=N / Y

number
Number of seconds or milliseconds during which the task is placed in the wait state, specified either as a decimal or a hexadecimal digit (X'....').

(1)
Specifies that register R1 is loaded with the required number of seconds or milliseconds.

Value range
- Specification in seconds:
  - direct entry: 0 ≤ number ≤ 4095 (=X'0FFF')
  - Register R1: 0 ≤ number ≤ 21599
- Specification in milliseconds: 1 ≤ number ≤ 999 (=X'3E7')

MSEC=
Defines the unit of measurement for the “number” specification.

N
Specification in seconds.

Y
Specification in milliseconds (ms).

CPUDEP=
Specifies whether the period of time is to be made relative to the CPU rate.

N
The time specification is not made relative.

Y
The time specification is not changed in the case of 1-MOPS servers (servers working at a rate of one million operations per second).
In the case of faster servers, the specified waiting time is reduced, relative to the CPU rate. With slower servers this time is increased correspondingly.

ℹ Making the time relative specification rate is advisable only when specified waiting times are short, since the number of executed instructions cannot otherwise be estimated precisely enough.

Notes on the macro call

- Time specifications ≤ 500 ms do not lead to deactivation of the task. The task remains in the active waiting state.
- The specification number = 0 sec corresponds to a waiting time of 500 ms (for reasons of compatibility).

For an **example** see the section “Contingency processes” on page 110.

VSVI1 – Output link and load information

General

Application area:	Linking and loading; see page 47
Macro type:	Type S, MF format **2**: standard/C/D/L/E/M form; see page 29

See also the “BLSSERV” manual [4] for information on the dynamic binder loader DBL.

Macro description

The **VSVI1** macro provides the user with information on entries in the DBL tables. For this purpose, DBL accesses the following contexts:

– user contexts and/or system contexts
  Information on system contexts is output only under TSOS.
– contexts of common memory pools in which shared code is stored and to which the user is connected.

The following information can be requested:

– a list of context names (SELECT=CTXLIST),
– the size of the code loaded in a context and the size of the associated link and load information (SELECT=CTXSIZE),
– a list containing the names, load addresses, lengths, types, attributes and contexts of CSECTs, ENTRYs and COMMONs (SELECT=ALLLIST),
– a list containing the names, load addresses, lengths, types, attributes and contexts of all CSECTs and COMMONs (SELECT=MODLIST),
– a record containing the name, load address, length, type, attribute, context, version and HSI code of a *single* control section (CSECT), ENTRY or COMMON (SELECT= BYNAME),
– a record containing the name, load address, length, type, attributes and context of a control section (CSECT) or COMMON specified by means of an address (SELECT= BYADDR).
– A list of ILEs belonging to one or more contexts.

The individual items of information (name, load address, length, attribute, type, context and HSI code) can be selected independently of one another. It is also possible to request just the length of the desired output information.
A description of the output information follows immediately after the operand description (see page 976).

Macro format and description of operands

VSVI1
SELECT=CTXLIST / CTXSIZE / ALLLIST / MODLIST / BYNAME / BYADDR / ILELIST ,OUTADDR=adr / (r) / label ,OUTLEN=integer ,ADDRESS=**Y**ES / **N**O ,CONTEXT=**Y**ES / **N**O ,CTXPRIV=ANY / YES / NO ,CTXSEL=ALL / LOCAL / GLOBAL / POOL / SSLOCAL ,HSI=**N**O / **Y**ES [,{ INNAME=name INNAME@=adr / (r) INADDR=adr / (r) / label }] [,{ INCTX=name INCTX@=adr / (r) }] ,INSTRUCT=adr ,INTVERS=BLSP2 / SRV001 / SRV002 / SRV003 ,LEN=**Y**ES / **N**O ,NAME=**Y**ES / **N**O ,RUNMOD=**S**TD / **A**DV ,SCOPE=ALL / USER_GROUP / GLOBAL / GROUP ,SIZONLY=**N**O / **Y**ES ,SYMTYP = **ANY** / CSECT / ISL / NOTISL ,TYPE=**Y**ES / **N**O ,VERSION=**N**O / **Y**ES ,MF=S / C / D / E / L / M [,PARAM=adr / (r)] ,PREFIX=P / p [,LABEL=name]

The operands are described in alphabetical order below.

ADDRESS=
Specifies whether the load addresses are to be included in the information output.

YES
The load addresses will be output.

NO
The load addresses will not be output.

CONTEXT=
Specifies whether the names of the contexts are to be included in the information output.

YES
The context names will be output.

NO
The context names will not be output.

CTXPRIV=
Specifies the access privilege for the context search (only in conjunction with RUNMOD=ADV).

– For nonprivileged users (not TSOS), the entry is ignored and internally set to CTXPRIV=NO.
– For privileged users (TSOS), the following applies:
If RUNMODE=ADV and the CTXPRIV parameter was not specified, DBL sets CTXPRIV=ANY.
If RUNMODE=STD, DBL ignores the input value for CTXPRIV and sets CTXPRIV=YES.

ANY
Both privileged and nonprivileged contexts are to be searched. For nonprivileged users, only nonprivileged contexts will be searched.

YES
Only *privileged* contexts will be searched. This option is permitted for privileged users only.

NO
Only *nonprivileged* contexts will be searched.

CTXSEL=
Specifies the scope of the context search (only in conjunction with RUNMOD=ADV).

- For nonprivileged users (not TSOS), only CTXSEL=LOCAL or CTXSEL=POOL is permitted. Any other entry is handled in the same way as CTXSEL=LOCAL.
- For privileged users (TSOS), the following applies:
  If RUNMODE=ADV and the CTXSEL parameter was not specified, DBL sets CTXSEL=ALL.
  If RUNMODE=STD, DBL ignores the input value for CTXSEL and sets CTXSEL=GLOBAL.

ALL
Contexts with the SYSTEM and USER scope will be searched. Contexts with the scope POOL and SSLOCAL will not be considered.

GLOBAL
Only contexts with the SYSTEM scope will be searched.

LOCAL
Contexts with the USER scope will be searched.
RUNMODE=ADV will also search the pre-loaded part of the context of subsystems if following conditions are true:
- the task in which the VSVI1 macro is called is linked to the subsystem
- the subsystem possesses the attribute MEMORY-CLASS=*BY-SLICE

POOL
Contexts of memory pools into which shared code was loaded by means of the **ASHARE** macro will be searched. The set of memory pool contexts can be restricted by specifying a memory pool scope (SCOPE operand).

SSLOCAL
Only local subsystem contexts will be searched.

HSI=
Specifies whether the output is to contain information on the hardware-software interface (only in conjunction with RUNMOD=ADV).

NO
The HSI code will not be output.

YES
The HSI code and HSI compiler information will be output. This operand value is relevant only for the output of symbol information and may be specified only in conjunction with RUNMOD=ADV.

INNAME=name
Specifies the name of a control section (CSECT), ENTRY or COMMON whose name, load address, length and attributes are to be output.“name” may be up to 32 characters long.

The INNAME operand must be specified in conjunction with the SELECT=BYNAME operand.

INNAME@=
May be specified only if MF=M. Specifies the address of a field containing the name of a control section (CSECT), ENTRY or COMMON.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

INADDR=
Specifies an address for which the corresponding items of information (name, load address, length and attributes) are to be output.
The INADDR operand must be specified in conjunction with the SELECT=BYADDR operand.

addr
Address of a field which contains the program address searched for.
May be specified only if MF=M.

(r)
r = register containing program address searched for. May be specified only if MF=M.

label
Program address. The address can be specified as a symbolic address or as a constant (X'...'). May be specified only if MF=S or MF=L.

INCTX=name
Specifies a context which is to be searched. “name” may be up to 32 characters long. If the operand is not used the contexts will be searched in accordance with the CTXSEL and CTXPRIV operand values; if CTXSEL=POOL is specified, the contexts will be searched in accordance with the SCOPE operand. The INCTX operand will be ignored if SELECT=CTXLIST has also been specified.

INCTX@=
May be specified only if MF=M.
Specifies the address of a field containing the name of the context which is to be searched.

addr
Address of an auxiliary field which contains the field address searched for.

(r)
r = register containing the field address searched for.

INSTRUCT=addr
This operand is only available if INTVERS=SRVxxx and xxx ≥ 001 is specified. It must be used with SELECT=BYNAME.
Symbolic address of a structure containing an EEN (extended external name). The structure consists of two 4-byte length fields, the first one containing the length of the extended external name, and the second one containing the address of the extended external name.

INTVERS=
This operand defines the version of the VSVI1 macro interface.

BLSP2
Default; corresponds to macro version 3.

SRV001
Corresponds to macro version 4. This version is supported as of BLSSERV V2.0.

SRV002
Corresponds to macro version 5. This version is supported as of BLSSERV V2.3B.

SRV003
Corresponds to macro version 6. This version is supported as of BLSSERV V2.5A.

MF=
For a general description of the MF operand, its operand values and any of the specified operands PARAM and PREFIX, see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A PREFIX can be specified in the C form, D form or M form of the macro
(see section “S-type macros” on page 29).

LABEL=name
May be specified only if MF=M.
Name of the structure, i.e. the DSECT which describes the operand list of the **VSVI1** macro. The operand is mandatory if there is no valid USING statement for the definition of the base address register for the DSECT of the parameter list.
The LABEL operand must be specified in conjunction with the PARAM operand. Both operands are used to produce a valid USING statement.

The following may be specified for “name”:

1. The name specified in the name field of a preceding macro `name VSVI1 MF=D`
2. The name “xVSVIDS” if no “name” has already been specified, where “x” is the value of the PREFIX operand of a preceding macro `VSVI1 MF=D, PREFIX=x`
   The default value for “x” is “P”.
3. The name of the longer DSECT containing the parameter list of the **VSVI1** macro if the macro `VSVI1 MF=C` was specified earlier.

LEN=
Specifies whether the lengths are to be included in the information output.

<u>YES</u>
The lengths will be output.

NO
The lengths will not be output.

NAME=
Specifies whether the names of CSECTs, ENTRYs and COMMONs are to be included in the information output.

<u>YES</u>
The names are to be output.

NO
The names are not to be output.

OUTADDR=
Specifies the address of a field into which DBL is to place the information.

addr
Address of an auxiliary field which contains the field address searched for.
May be specified only if MF=M.

(r)
r = register containing the field address searched for. May be specified only if MF=M.

label
Symbolic field address.May be specified only if MF=S or MF=L.

OUTLEN=integer
Specifies the length of the output field (in bytes). The minimum length is 4 bytes.

RUNMOD=
Specifies the operating mode in which the **VSVI1** macro will be processed.

<u>STD</u>
The macro will be processed in the standard form. The information output in this case is fully compatible with the information supplied by the previous VSVI macro of BS2000 Version 9.5.
In standard form the **VSVI1** macro processes only 8-character names in the case of CSECTs, ENTRYs and COMMONs, and only names with up to 16 characters in the case of contexts. This can lead to conflicts if, for example, information on a loaded LLM is to be output, since longer names are truncated.

ADV
The macro will be processed in the extended form. In this form the **VSVI1** macro processes names consisting of up to 32 characters. The following operands are permitted only in conjunction with RUNMOD=ADV:
SELECT=ILELIST, CTXSEL, SCOPE, CTXPRIV, HSI, VERSION.

SCOPE=
Specifies which memory pool contexts are to be considered if CTXSEL=POOL was specified (only in conjunction with RUNMOD=ADV).

ALL
Memory pool contexts will be considered.

GROUP
Only the contexts of common memory pools with the GROUP scope will be considered.

USER_GROUP
Only the contexts of common memory pools with the USER_GROUP scope will be considered.

GLOBAL
Only the contexts of common memory pools with the GLOBAL scope will be considered.

SELECT=
Selects the type of information to be output.

CTXLIST
A list of context names will be output (user (contexts of the task and/or system contexts and/or memory pool contexts and/or local subsystem contexts).

CTXSIZE
Only allowed in combination with RUNMOD=ADV.
The size of the code loaded into a context and the size of the associated link and load information will be output. The size is specified in bytes and is rounded up to a multiple of 4 Kbytes. The name of the context must be defined using INCTX or INCTX@.

ALLLIST
A list containing the names, load addresses, lengths and attributes of all CSECTs, ENTRYs and COMMONs will be output.

MODLIST
A list containing the names, load addresses, lengths and attributes of all CSECTs and COMMONs will be output.

BYNAME
A record containing the name, load address, length and attributes of a *single* control section (CSECT), ENTRY or COMMON will be output. The name of the symbol must be specified with the INNAME or INSTRUCT (and INTVERS=SRVxxx with xxx ≥ 001) operand.

BYADDR
A record containing the name, load address, length and attributes of a *single* control section (CSECT) or COMMON will be output. The address of the symbol must be specified with the INADDR operand.

ILELIST
A list containing information on ILEs in the specified context is to be output. SELECT=ILELIST may be specified only in conjunction with RUNMOD=ADV.

SIZONLY=
Specifies whether the desired information or merely the length of the desired information will be output.

NO
The information will be transferred to the output field.

YES
Only the length of the information, not the information itself, will be transferred to the output field.

SYMTYP=
This operand is available when INTVERS=SRVxxx and xxx ≥ 003 are specified. It is only meaningful in conjunction with the SELECT=BYNAME operand, and is ignored in any other case.
It specifies the type of the symbol defined with INNAME, INNAME@ or INSTRUCT for which information is requested.

ANY
The type of the symbol searched for is irrelevant.

CSECT
Only CSECTs with the specified name are searched for.

ISL
This specification is only relevant for privileged users. Furthermore, it is also only meaningful if the CP context (see INCTX/INCTX@) or privileged contexts (CTXSEL=ALL/GLOBAL and CTXPRIV=ANY/ALL) are searched.

Only ISL ENTRYs with the specified name are searched for.

NOTISL
Only symbols with the specified name are searched for, the CSECTs or ENTRYs, but not ISL.

TYPE=
Specifies whether the types (CSECT/ENTRY/COMMON) are to be included in the information output.

YES
The types will be output.

NO
The types will not be output.

VERSION=
Specifies whether the output is to contain information on the program version (only in conjunction with RUNMOD=ADV).

NO
The program version will not be output.

YES
The program version will be output. This operand value is relevant only for the output of symbol information and may be specified only in conjunction with RUNMOD=ADV.

Notes on the macro call

- At least one of the operands NAME, ADDRESS, LEN, TYPE, CONTEXT, VERSION or HSI must be specified for the output of information.
- None of the operands NAME, ADDRESS, LEN, TYPE, CONTEXT, VERSION or HSI needs to be specified if the user merely wishes to check whether
  - a specified name (INNAME/INSTRUCT operand) is the name of a symbol (SELECT=BYNAME operand),
  - a specified address (INADDR operand) is available (SELECT=BYADDR operand),
  - a specified context (INCTX operand) is available (SELECT=ALLLIST or SELECT=MODLIST operand).

  In these cases, only the return code is significant.

  If the name, address or context cannot be found the following return code is passed:

X'0440003C'	Name of symbol not found
X'04400038'	Address not found
X'04400040'	Context not found

  If the name, address or context is found DBL delivers the return code X'00000000'.
- The length of the output area into which DBL is to transfer the information has to be defined with the OUTLEN operand. If the specified length is too small the information will be truncated to this length. If the specified length is less than the length of the smallest item of information, no information will be output. In both cases DBL supplies a return code.

- If a module contains more than one CSECT, the information selected by DBL is dependent on the SELECT operand, as follows:
  - with SELECT=BYADDR the information refers to the CSECT containing the address specified by INADDR.
  - with SELECT=BYNAME, and if INNAME identifies a CSECT name, the information refers to this CSECT.
  - with SELECT=ALLLIST separate information is output for each CSECT.
- The **VSVI1** macro attempts to carry on processing as far as possible in the event of an error. If, for example, the operands specify that more than one context has to be searched, and an error occurs while a context is being searched, the **VSVI1** macro continues by searching the next context. The cause of the error is reported by DBL in a return code. This return code always refers to the last error that occurred.
- If SELECT=BYNAME is specified, either the INNAME or INSTRUCT operand must be specified. However, they cannot be specified together.
- When SELECT=BYNAME is set, DBL searches the context only until the *first* symbol of the specified name (INNAME or INSTRUCT operand) is found. If there are several symbols with the same name, only information on the first symbol found is output.
- If just the length of the information is desired (SIZONLY=YES operand), DBL takes the length of the information up to the null entry as the length.
- If the memory area specified in OUTADDR is only readable, or if zero was specified for OUTLEN, processing is aborted with a user dump.
- If a context list is requested (SELECT=CTXLIST), NO can be specified for the NAME, ADDRESS, LEN, TYPE, CONTEXT, VERSION, HSI and SIZONLY operands.
- When information about contexts of common memory pools is being requested (CTXSEL=POOL), the return code X'08400048' may be issued as a result of concurrent access to a common memory pool.
- The following applies for nonprivileged users (not $TSOS):
  - The user-specific values for the operands are ignored and are filled internally by the DBL with CTXPRIV=NO.
  - For the CTXSEL operand, values other than POOL are ignored and are filled internally by the DBL with CTXSEL=LOCAL.
  - If VSVI1 is called SELECT=BYNAME but without INCTX, the DBL searches first in the private class 6 memory and in common memory pools. If it does not find the symbol specified, it tries to establish a connection to nonprivileged subsystems.

Possible operand combinations

The following table shows which operands are mandatory or permitted depending on the SELECT operand:

SELECT=	INADDR	INNAME/ INSTRUCT	INCTX	OUTADDR	OUTLEN	CTXSEL	CTXPRIV	SCOPE	SYMTYP
CTXLIST	-	-	-	M	M	P	P	P	I
CTXSIZE	-	-	M	M	M	-	-	-	I
ALLLIST	-	-	P	M	M	P	P	P	I
MODLIST	-	-	P	M	M	P	P	P	I
ILELIST	-	-	P	M	M	P	P	P	I
BYADDR	M	-	P	M	M	P	P	P	I
BYNAME	-	M	P	M	M	P	P	P	P

M	Parameter mandatory
P	Operand permitted
I	Operand ignored
-	Operand irrelevant for this SELECT specification.

Output information when RUNMOD=STD

In RUNMOD=STD operating mode the macro is processed in the standard form. The information output in this case is fully compatible with the information supplied by the previous VSVI macro of BS2000 Version 9.5.

In standard form the **VSVI1** macro processes only 8-character names in the case of CSECTs, ENTRYs and COMMONs, and only names with up to 16 characters in the case of contexts. Longer names of CSECTs, ENTRYs and COMMONs are truncated to 8 characters, and longer names of contexts to 16 characters.

The reference size for an item of information output is an entry in the DBL tables. The entry has a *fixed* length of 36 bytes.

Format of the entries

Byte	Length	Field	Field entry	Coding/Remarks
0- 7	8	Name of symbol	left-justified with trailing blanks	Character constant
8-11	4	Load address	-	Hexadecimal constant
12-15	4	Length	-	Hexadecimal constant
16	1	Type	-	X'F0' ≙ CSECT X'F1' ≙ ENTRY X'F3' ≙ COMMON
17	1	Attributes	1 or 2 bit per attribute	2^7 ≙ INVISIBILITY 2^6 and 2^5 ≙ AMODE 00 ≙ AMODE=32 01 ≙ AMODE=31 10 ≙ AMODE=24 11 ≙ AMODE=ANY 2^4 ≙ RESIDENT 2^3 ≙ PAGE 2^2 ≙ READ-ONLY
18-19	2		X' 0000'	Alignment of the following field
20-35	16	Name of the context	left-justified with trailing blanks	Character constant

If just the length of the information is desired (SIZONLY=YES operand), this takes up the first 4 bytes of the output field.

Multiple entries are output consecutively in the same sequence. The last entry is followed by a null entry. A pseudoentry is output if the specified program address cannot be found (SELECT=BYADDR operand).

Field	Null entry		Pseudoentry
	Length	Field entry	
Name of CSECT/COMMON/ENTRY	8	X'40.......40'	C'ABSOLUTE'
Load address	4	X'0.....0'	X'0.....0'
Length	4	X'F.....F'	X'0.....0'
Type	1	X'C5'	X'00'
Attribute	1	X'00'	X'00'
Name of context	16	X'40.......40'	X'40..........40'

Notes

- The length of the output field is derived from adding together the individual entries. If the length is too small the information will be truncated. The information is not transferred if it is shorter than the minimum length for a requested entry. The minimum length = length of an entry minus the length of items that are not to be output (e.g. NAME=NO).
- Only the INVISIBILITY and AMODE attributes are relevant for an ENTRY, and only PAGE and AMODE for a COMMON.
- A control section or ENTRY has the INVISIBILITY attribute if it was masked during linking or subsequent loading.
- If SELECT=CTXLIST is specified, a list of 16-byte context names is output, each followed by 16 blanks (X'40') to indicate the end of the name.

  Format:

CTX1␣␣␣␣␣␣␣␣␣␣␣␣
CTX2␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣

- If SELECT=CTXSIZE is specified, the output information consists of two words. The first word contains the size of the code in the context and the second word contains the size of the link and load information for this context. The specification is given in bytes and is rounded up to a multiple of 4 Kbytes.

Output information when RUNMOD=ADV

In RUNMOD=ADV operating mode the macro is processed in the extended form. In extended form the macro processes names up to 32 characters long, and ILE information, HSI code and HSI compiler information may be requested in addition. The reference size for a specified item of information is an entry in the DBL tables. The entry is of *variable* length, being dependent on the length of the names and the type of information requested.

Format of the entries

Bytes	Length	Field	Field entry	Coding/comment
0 - 3	4	Load address	—	Hexadecimal constant
4 - 7	4	Length	—	Hexadecimal constant
8	1	Type	—	X'F0' = CSECT X'F1' = ENTRY X'F2' = COMMON
9	1	Attributes	1 or 2 bit per attribute	2^7 ≙ INVISIBILITY 2^6 and 2^5 ≙ AMODE 00 ≙ AMODE=32 01 ≙ AMODE=31 10 ≙ AMODE=24 11 ≙ AMODE=ANY 2^4 ≙ RESIDENT 2^3 ≙ PAGE 2^2 ≙ READ-ONLY
10 - 11	2		X'0000'	Alignment of following field
12	1	HSI code	—	X'01' = 7500 (/390) X'09' = x86
13	1	HSI compiler information	—	Compiler-dependent; BLS simply records this information and outputs it here.
14	1	Length n of the symbol name	—	Hexadecimal constant
15	n	Name of the symbol	—	Alphanumeric constant
15+n	1	Length m of the version	—	Hexadecimal constant
15+(n+1)	m	Version of the program to which the symbol belongs	—	Alphanumeric constant
15+(n+1) +m	1	Length l of the context name	—	Hexadecimal constant
15+(n+1) +(m+1)	l	Name of the context	—	Alphanumeric constant

The total length of an entry may be calculated as follows:

Entry length = 17 + n + m + l

If just the length of the information is desired (SIZONLY=YES operand), this takes up the first 4 bytes of the output field.

Blank entries and pseudoentries

Multiple entries are output consecutively in the same order. The last entry is followed by a null entry. A pseudoentry is output if the specified program address cannot be found (SELECT=BYADDR operand).

Field	**Null entry**		**Pseudoentry**
	Length	**Field entry**	
Load address	4	X'0.....0'	X'0.....0'
Length	4	X'F.....F'	X'0.....0'
Type	1	X'C5'	X'00'
Attribute	1	X'00'	X'00'
Length n of symbol name	1	X'08'	X'08'
Name of symbol	8	X'40......40'	C'ABSOLUTE'
Length m of context name	1	X'00'	X'00'

Notes

- If SELECT=CTXLIST is specified, a list of variable-length context names is output. The first byte of each list item contains the length of the context name; the following bytes contain the context name proper. The list is terminated with a list item containing 32 blanks (X'40').

  Format:

0D	LOCAL#DEFAULT
04	CTX1
04	CTX2
20	␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣

- If SELECT=CTXSIZE is specified, the output information consists of two words. The first word contains the size of the code in the context and the second word contains the size of the link and load information for this context. The specification is given in bytes and is rounded up to a multiple of 4 Kbytes.
- If a user (not $TSOS) calls the **VSVI1** macro with SELECT=BYNAME but without specifying INCTX, and if DBL cannot find this symbol either in this user's class 6 memory or in shared code in memory pools, DBL attempts to establish a connection to the nonprivileged DSSM subsystems.

- If SELECT=ILELIST is specified, a formatted list entry is output for each ILE. The context name may also be included in this list on request. If CONTEXT=NO is specified the context name is not output. An entry for an ILE symbol has the following format:

Bytes	**Length**	**Field name**	**Meaning and/or values**
0	1	STATE	X'01' = ACTIVE X'02' = NOT_ACTIVE
1	1	CONTROL	X'01' = SYSTEM X'02' = USER
2	1	HSI_CODE	X'01' = 390 X'09' = x86
3	1	RESERVED1	Reserved (must contain X'00')
4	4	LOAD_ADDR	Address of the IL routine
8	4	SERVER_ADDR	Address of the ILE server
12	2	REF_DISPL	Distance of the external reference to the server within the IL routine
14	32	NAME	Name of the ILE symbol
46	2	RESERVED2	Reserved (must contain X'0000')
48	32	CONTEXT	Name of the context to which the ILE belongs

The entry format roughly corresponds to the format that is also generated when the **ILEMIT** macro is called.

The last ILE entry is followed by a blank entry in which all fields (except the NAME field) contain binary zeros. The NAME field contains blanks. A blank entry does not contain a CONTEXT field.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the VSVI1 macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	The macro was executed normally.
X'0C'	X'01'	X'0018'	A reserved field in the parameter list is not preset with zeros.
X'0C'	X'01'	X'0020'	The specified length of the output field (OUTLEN operand) is less than the actual length of the field.
X'0C'	X'01'	X'0024'	The output field (OUTADDR operand) is not aligned on a halfword boundary, has read access only, or has not been assigned.
X'0C'	X'01'	X'0028'	Illegal entry for the SELECT operand.
X'0C'	X'01'	X'002C'	Illegal entry for an operand. This may be: – HSI or VERSION was specified in conjunction with RUNMOD=STD, – (syntactically) wrong name specified for INNAME, INSTRUCT or INCTX – privileged context specified and the user is nonprivileged – a nonprivileged user has specified an illegal context name in RUNMOD=STD operating mode. When RUNMOD=STD applies, only information on contexts with names consisting of blanks (X'40') or the character string “LOCAL#DEFAULT” may be requested by a nonprivileged user. – SELECT=BYADDR was specified without INADDR or with incorrect value for INADDR (e.g. INADDR=X'FFFFFFFF'). – a nonprivileged user has specified SYMTYP with a value other than ANY and `SELECT=BYNAME,CTXSEL=LOC,RUNMOD=ADV`.
X'0C'	X'01'	X'002D'	Illegal entry for the SYMTYP operand.
X'0C'	X'01'	X'0030'	The caller has not specified any of the information operands (NAME, ADDRESS, LEN, TYPE, CONTEXT, VERSION, HSI) and SELECT is not BY_NAME or BY_ADDR. This return code is also transferred if: – the operands NAME, ADDRESS, LEN, TYPE, HSI, CONTEXT and VERSION all contain NO, and SIZONLY=YES was specified. – the operands NAME, ADDRESS, LEN, TYPE, HSI, CONTEXT and VERSION all contain NO, SIZONLY=NO, SELECT=MODLIST or SELECT=ALLLIST was specified, and INCTX was not specified.
X'08'	X'40'	X'0034'	The length of the output area specified in the OUTLEN operand is less than the total length of the requested information. The output is incomplete.

X'cc'	X'bb'	X'aaaa'	Meaning
X'0C'	X'01'	X'0034'	The length of the output area specified in the OUTLEN operand is too small for transferring the smallest item of information requested. No output takes place.
X'04'	X'40'	X'0038'	The address specified in the INADDR operand belongs to none of the modules already loaded.
X'04'	X'40'	X'003C'	The name specified in the INNAME or INSTRUCT operand designates none of the modules already loaded.
X'04'	X'40'	X'0040'	The context name specified in the INCTX operand cannot be found or the task from which the VSVI1 call is issued is not linked to the subsystem with the specified context name.
X'0C'	X'20'	X'0044'	Internal error while the function was being executed. No output is possible.
X'08'	X'40'	X'0048'	One or more contexts with global scope are being used by another task. The output is incomplete.
X'04'	X'40'	X'004C'	The name of the symbol is longer than 8 characters and RUNMOD=STD is specified. The name has been truncated to 8 characters.
X'04'	X'40'	X'0050'	The specified context, though present, is empty because all its objects have been unloaded.
X'04'	X'40'	X'0070'	The task is not connected to the memory pool.
X'0C'	X'20'	X'0198'	No more memory available.
X'0C'	X'40'	X'0204'	Internal error in memory management.
X'0C'	X'40'	X'0208'	Internal error in data manager.
X'0C'	X'40'	X'020C'	Internal error in the symbolic information tables.
X'0C'	X'20'	X'0300'	Error in $REQM, $RELM (system error).
X'00'	X'01'	X'FFFF'	The function is no longer or not yet supported.
X'00'	X'03'	X'FFFF'	The interface version is not supported.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard return codes” on page 43.

Example

During the PROGA program run, a **BIND** macro is issued to load a second control section, PROGB, as an overlay. PROGB is stored as an object module in library MACEXMP.LIB. Before and after PROGB is loaded, the **VSVI1** macro is called in order to transfer link and load information from the DBL tables to an output area. Both control sections are to execute in 31-bit addressing mode. PROGA is to be loaded below, and PROGB above, the 16-MB boundary. An ENTRY is defined in PROGB. After the **BIND** macro is called, PROGB is to execute first. Following the execution of PROGB a return branch is to be made to PROGA.

Source listing

```
PROGA    START
PROGA    AMODE 31                                                  (1)
PROGA    RMODE 24
         BALR  3,0
         USING *,3
         USING BINDDS,6                                            (2)
         USING VSVI1DS,7                                           (3)
         ST    3,AREA11
         UNPK  AREAH,AREA1
         MVC   AREAA(8),AREAH
WROUT1   WROUT OUT,ERROR,PARMOD=31                                 (4)
         MVI   ADR1,X'D1'                                          (5)
         MVC   ADR1+1(L'ADR1-1),ADR1
         VSVI1 MF=E,PARAM=VSVI1PAR                                 (6)
         LA    7,VSVI1PAR
         CLC   YVSVRET,=X'00000000'                                (7)
         BNE   VSVIERR                                             (8)
BACK     LA    12,VSVI
BIND     BIND  MF=E,PARAM=BINDPAR                                  (9)
         LA    6,BINDPAR
         CLC   XBINRET,=X'00000000'                               (10)
         BE    VSVI
         MVC   OUT+5(28),='BIND ERROR!                 '
         WROUT OUT,ERROR,PARMOD=31                                (11)
         B     ERROR
VSVI     VSVI1 MF=E,PARAM=VSVI1PAR                                (12)
         CLC   YVSVRET,=X'00000000'                               (13)
         BE    MVC
VSVIERR  MVC   OUT+5(28),='VSVI1 ERROR!                '
         WROUT OUT,ERROR,PARMOD=31                                (14)
         B     ERROR
MVC      MVC   OUT+5(28),='VSVI1 PROCESSED             '
         WROUT OUT,ERROR,PARMOD=31                                (15)
         MVC   OUT+5(28),='RETURN TO PROGA             '
         WROUT OUT,ERROR,PARMOD=31
```

```
ERROR    TERM
****************
         DS    0F
ADR1     DS    CL180 ---------------------------------------------------- (16)
OUT      DC    Y(OUTE-OUT)
         DS    CL3
         DC    C'PROGA: BASE REG.= '
AREAA    DS    CL8
OUTE     EQU   *
AREA     DS    0F
AREA1    DS    0CL5
AREA11   DS    CL4
AREA12   DC    C'0'
         DS    0F
AREAH    DS    CL9
BINDPAR  BIND  MF=L,SYMBOL=PROGB,SYMBLAD=PROGB@,BRANCH=YES,PROGMOD=ANY,*
               LIBLINK=PLAMLIB ------------------------------------------ (9)
VSVI1PAR VSVI1 MF=L,SELECT=ALLLIST,CTXSEL=ALL,OUTADDR=ADR1,OUTLEN=180 --- (6)
PROGB@   DS    A
BINDDS   BIND  MF=D,PREFIX=X -------------------------------------------- (17)
VSVI1DS  VSVI1 MF=D,PREFIX=Y -------------------------------------------- (18)
         END
PROGB    CSECT PAGE ----------------------------------------------------- (19)
PROGB    AMODE ANY
PROGB    RMODE ANY
         ENTRY ENTR ----------------------------------------------------- (20)
ENTR     BALR  4,0
         USING *,4
         ST    4,AREA11
         UNPK  AREAH,AREA1
         MVC   AREAA(8),AREAH
         WROUT OUT,ERROR,PARMOD=31 -------------------------------------- (21)
         BR    12
ERROR    TERM
****************
OUT      DC    Y(OUTE-OUT)
         DS    CL3
         DC    C'PROGB: BASE REG.= '
AREAA    DS    CL8
OUTE     EQU   *
AREA     DS    0F
AREA1    DS    0CL5
AREA11   DS    CL4
AREA12   DC    C'0'
AREAH    DS    CL9
         END
```

(1) The attribute AMODE=31 is defined for control section PROGA. The attribute RMODE=24 means that PROGA will always be loaded below the 16-MB boundary.

(2) Register 6 is assigned to the assembler as the base address register for addressing the DSECT for the operand list of the **BIND** macro, which is generated at the symbolic address BINDDS as a result of a **BIND** macro specifying MF=D.

(3) Register 7 is assigned to the assembler as the base address register for addressing the DSECT for the operand list of the **VSVI1** macro, which is generated at the symbolic address VSVI1DS as a result of a **VSVI1** macro specifying MF=D.

(4) The contents of the base register for PROGA are output to indicate the addressing mode and the load address.

(5) The output field for the **VSVI1** macro is preset to C'J'.

(6) The **VSVI1** macro is called in its E form. At this point in the program, therefore, only the instruction code is generated. The associated operand list is created at the symbolic address VSVI1PAR by means of a **VSVI1** macro specifying MF=L. As a result of the operand values specified in the list, the **VSVI1** macro causes a list containing the names, load addresses, lengths and attributes of all CSECTs, ENTRYs and COMMONs to be output before PROGB is loaded.

(7) Following execution of the **VSVI1** macro, a check is made to verify that the YVSVRET field of the standard header contains the return code X'00000000', which indicates error-free execution of the macro. The name YVSVRET originates from the DSECT that was generated under the symbolic address VSVI1DS as a result of a **VSVI1** macro specifying MF=D and PREFIX=Y (see (18)). This DSECT describes the layout of the operand list of the **VSVI1** macro. The symbolic names of the DSECT can be used for addressing within the operand list once the assigned base address register (in this case, register 7) has been loaded with the start address of the operand list (in this case, VSVI1PAR).

(8) If the **VSVI1** macro does not execute without error, a branch is made to error exit VSVIERR, an error message is output to SYSOUT and the PROGA program run is terminated.

(9) The **BIND** macro is called in its E form at the symbolic address BIND. At this point in the program, therefore, only the instruction code is generated. The associated operand list is created at the symbolic address BINDPAR by means of a **BIND** macro specifying MF=L. As a result of the operand values specified in the list, the **BIND** macro causes the following to happen at program runtime:

- the CSECT PROGB (SYMBOL=PROGB) is reloaded from the library assigned with the link name PLAMLIB (LIBLINK=PLAMLIB)
- the start address of PROGB is stored in field PROGB@ (SYMBLAD=PROGB@)

- the 31-bit addressing mode is set for PROGB (PROGMOD=ANY)
- the program run is continued in PROGB after PROGB is loaded (BRANCH=YES).

(10) Following execution of the **BIND** macro, a check is made to verify that the XBINRET field of the standard header contains the return code X'00000000', which indicates error-free execution of the macro. The name XBINRET originates from the DSECT that was generated under the symbolic address BINDDS as a result of a **BIND** macro specifying MF=D and PREFIX=Y (see (17)). This DSECT describes the layout of the operand list of the **BIND** macro. The symbolic names of the DSECT can be used for addressing within the operand list once the assigned base address register (in this case, register 6) has been loaded with the start address of the operand list (in this case, BINDPAR).

(11) If the **BIND** macro does not execute without error, an error message is output to SYSOUT and the PROGA program run is terminated.

(12) Same as (6), but after PROGB has been loaded.

(13) Same as (7), but after PROGB has been loaded.

(14) If the **VSVI1** macro does not execute without error, an error message is output to SYSOUT and the PROGA program run is terminated.

(15) Messages to SYSOUT indicate that the program run has been continued in PROGA and then link and load information has been output by means of the **VSVI1** macro.

(16) Output field for the **VSVI1** macro.

(17) The **BIND** macro with MF=D generates a DSECT which describes the layout of the operand list of the **BIND** macro. The operand PREFIX=X causes the letter X to be prefixed to all symbolic names in this DSECT (field names and equates).

(18) The **VSVI1** macro with MF=D generates a DSECT which describes the layout of the operand list of the **VSVI1** macro. The operand PREFIX=Y causes the letter Y to be prefixed to all symbolic names in this DSECT (field names and equates).

(19) The CSECT statement defines the control section PROGB with the attributes AMODE=ANY and PAGE.

(20) ENTRY statement for the symbolic address ENTR.

(21) The contents of the base register for PROGB are output to indicate the addressing mode and the load address.

Runtime listing

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,proga), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,proga)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 786 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 232 MSEC
//compile source=*library-element(macexmp.lib,progb), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,progb)), -
//        test-support=*aid
%  ASS6011 ASSEMBLY TIME: 191 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 93 MSEC
//end
%  ASS6012 END OF ASSEMBH
/add-file-link link-name=plamlib,file-name=macexmp.lib ────────────────  (1)
/load-executable-program library=macexmp.lib,element-or-symbol=proga -    (2)
//        program-mode=*any,test-options=*aid
%  BLS0523 ELEMENT 'PROGA', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'PROGA', VERSION ' ' OF '<date> <time>' LOADED
/%in back;%in error;%r ───────────────────────────────────────────────  (3)
PROGA: BASE REG.= 80000002 ───────────────────────────────────────────  (4)
STOPPED AT LABEL: BACK , SRC_REF: 53, SOURCE: PROGA , PROC: PROGA
```

```
/%d adr1 %x ——————————————————————————————————————————————————————— (5)
*** TID: 00100136 *** TSN: 1E17 *********************************************
CURRENT PC: 00000054    CSECT: PROGA  ***************************************
V'00000144' = ADR1     + #'00000000'
00000144 (00000000) D7D9D6C7 C1404040 00000000 00000384    PROGA   .......d
00000154 (00000010) F0200000 D3D6C3C1 D37BC4C5 C6C1E4D3    0...LOCAL#DEFAUL
00000164 (00000020) E3404040 40404040 40404040 00000000    T           ....
00000174 (00000030) FFFFFFFF C5000000 40404040 40404040    ~~~~E...
00000184 (00000040) 40404040 40404040 D1D1D1D1 D1D1D1D1            JJJJJJJJ
00000194 (00000050) D1D1D1D1 D1D1D1D1 D1D1D1D1 D1D1D1D1    JJJJJJJJJJJJJJJJ
           REPEATED LINES:     4
000001E4 (000000A0) D1D1D1D1 D1D1D1D1 D1D1D1D1 D1D1D1D1    JJJJJJJJJJJJJJJJ
000001F4 (000000B0) D1D1D1D1                               JJJJ
/%r
PROGB: BASE REG.= 81000002 ———————————————————————————————————————— (6)
VSVI1 PROCESSED
RETURN TO PROGA
STOPPED AT LABEL: ERROR , SRC_REF: 208, SOURCE: PROGA , PROC: PROGA
/%d adr1 %x ——————————————————————————————————————————————————————— (7)
CURRENT PC: 00000126    CSECT: PROGA  ***************************************
V'00000144' = ADR1     + #'00000000'
00000144 (00000000) D7D9D6C7 C2404040 01000000 0000008A    PROGB   ........
00000154 (00000010) F0680000 D3D6C3C1 D37BC4C5 C6C1E4D3    0...LOCAL#DEFAUL
00000164 (00000020) E3404040 C5D5E3D9 40404040 01000000    T   ENTR    ....
00000174 (00000030) 00000000 F1600000 D3D6C3C1 D37BC4C5    ....1-..LOCAL#DE
00000184 (00000040) C6C1E4D3 E3404040 D7D9D6C7 C1404040    FAULT   PROGA
00000194 (00000050) 00000000 00000384 F0200000 D3D6C3C1    .......d0...LOCA
000001A4 (00000060) D37BC4C5 C6C1E4D3 E3404040 40404040    L#DEFAULT
000001B4 (00000070) 40404040 00000000 FFFFFFFF C5000000        ....~~~~E...
000001C4 (00000080) 40404040 40404040 40404040 40404040
000001D4 (00000090) D1D1D1D1 D1D1D1D1 D1D1D1D1 D1D1D1D1    JJJJJJJJJJJJJJJJ
000001E4 (000000A0) D1D1D1D1 D1D1D1D1 D1D1D1D1 D1D1D1D1    JJJJJJJJJJJJJJJJ
000001F4 (000000B0) D1D1D1D1                               JJJJ
```

(1) The file link name used in the **BIND** call of program PROGA is assigned.

(2) DBL is invoked to link and load the program.

(3) The AID command %INSERT is used to define the test points BACK and ERROR. The %RESUME command passes control to the called program.

(4) The contents of the base register for PROGA are output. 31-bit addressing is set (bit 2^{31} = 1); the load address is below the 16-MB boundary.

(5) The link and load information has been transferred to field ADR1. ADR1 was preset to X'D1'. The **VSVI1** macro call was issued before PROGB was loaded. The first 8 bytes reveal the name PROGA for the first control section. It is followed by the load address X'00000000' and the length X'0000037C' (892 bytes). The next two bytes indicate the type and attribute of the control section. Type X'F0' (≙ CSECT) and attribute X'20' (≙AMODE=31). The following values (X'0000') are used for alignment. Then comes the 16-byte context name (LOCAL#DEFAULT). The following fields contain the “null entry” for name (X'40....40'), load address (X'00000000'), length (X'FFFFFFFF'), attribute (X'C5') and context name (X'40....40').

(6) After PROGB has been loaded, the program run is continued in PROGB. The contents of the base register are output. 31-bit addressing mode is set. The load address is located above 16 MB. After PROGB has been completed, PROGA is continued.

(7) Following loading of PROGB, there are several DBL entries in ADR1. First is the entry for PROGB and the ENTRY in PROGB, then the entry for PROGA. The load address for PROGB is X'01000000', the length X'00000082' (130 bytes). Its type is X'F0' (CSECT) and its attributes X'68' (AMODE=ANY and PAGE). The ENTRY name is C'ENTR', type X'F1' (ENTRY), attribute X'60' (AMODE=ANY). Then comes the 16-byte context name (LOCAL#DEFAULT). This is followed by the entry for PROGA and finally by the null entry.

VTCSET – Define logical control characters

General

Application areas:	Requesting and accessing lists and tables; see page 155 Data terminal communication; see page 160
Macro type:	Type O; see page 28

- This macro description applies to VTSU V13.3A

Macro description

The **VTCSET** macro generates symbolic names which can be used to insert logical control characters in line mode output messages and to locate line mode inputs.

Macro format and description of operands

VTCSET
prefix

prefix
Specifies a string (up to 5 characters) to be prefixed to the symbolic names.

In the following descriptions, “&P.” is used as a prefix in place of the string which precedes the symbolic names.

1. Logical record control characters

&P.NL
Logical end of line (new line)

Effect on output:

– Special display formats on the terminal are reset to the standard display format (normal, standard color, unprotected in line mode, protected in extended line mode). The character set is not reset. If the message is output in extended line mode, the remainder of the line is blanked and protected.

– The defined logical end-of-line character is output (except in extended line mode).

– The next line is set to standard status: normal, low intensity, standard color, standard character set, unprotected in line mode, protected in extended line mode. The standard character set is character set 0 for 9763 Data Display Terminals and character set 1 for other display terminals and printers.

Status is not set to standard if operating mode 2 and HOM=YES were specified for the terminal.

- If continuing the output would cause data overflow at the terminal, the defined overflow check is executed (except in extended line mode).
- The cursor is positioned at the start of the next line.
- In the case of field-oriented display and structured output (standard) a start of field is generated.

&P.NP
Logical end of page (new page)

Effect on output:

- Special terminal display formats are reset to the standard display format (normal, standard character set, standard color, unprotected in line mode, protected in extended line mode with UPDATE=NO, unprotected in extended line mode with UPDATE=YES). The screen format is reset to 24x80. The standard character set is character set 0 for 9763 Data Display Terminals and character set 1 for other display terminals and printers.
- The defined logical end-of-line character is output.
- Hardcopy output is initiated (if HCOPY=YES).
- The defined overflow control action is executed.
- A new page is created (on display terminals the screen is cleared and the screen format 24x80 set, on printers a page feed is performed). If page feeds were already initiated in the same message by means of ASF (automatic sheet feed), then the last ASF used replaces NP.
- The cursor is positioned at start of line.
- In the case of field-oriented display and structured output (standard) a start of field is generated.

&P.CL
Logical end of record (current line)

Effect on output (printers and teleprinters only):

- Special display formats on the terminal are reset (to normal, standard character set, unprotected). The standard character set is character set 1 for the devices concerned).
- The defined logical end-of-line character is output.
- The cursor is set to the start of the current line.

&P.VPAddd
(data display terminals only, see page 1005 for printers) Position on first unprotected field of a line (vertical position absolute) (binary or three-digit decimal value)

Effect on output:
Absolute line positioning on the first unprotected field in line ddd of the data display terminal. Can be combined with HPA for simultaneous positioning on the absolute column.

Note

- If ddd is zero or larger than 255, the substitute character (SUB) is inserted instead of VPA and ddd is output.
- If ddd is less than or equal to 255 but greater than the maximum number of lines, NL is executed instead of VPA.

&P.HPA ddd
(data display terminals only, see page 1005 for printers) Position on column (horizontal position absolute) (three-digit decimal value)

HPA is processed in extended line mode only and only when specified immediately after a valid VPA. HPA specifies the absolute column in a line defined by the preceding VPA. If the preceding VPA control character was invalid or if HPA is not specified immediately after VPA, HPA ddd is ignored. HPA on a 3270 terminal positions on the start of the next unprotected field.

Effect on output:
Absolute column positioning in the line defined by the preceding VPA.

Note

- If ddd is zero or larger than 255, the substitute character (SUB) is inserted instead of HPA and ddd is output.
- If ddd is less than or equal to 255 but greater than the maximum number of columns, HPA ddd is ignored and only the VPA function is executed.

2. Logical display control characters

&P.EM1
Emphasized layout 1

Effect on output:
Subsequent text characters are “emphasized” (highlighted) on the terminal as appropriate to the device type used (see table on page 1013).

Emphasized layout is reset by:
– the logical record control characters (NL, NP, HPA, VPA)
– the display control characters (EM2, EM3, EM4, DAR, DIS, NOR)
– the field control characters (EPA, SPA, NUM, CHS, COL, FLD)

&P.EM2
Emphasized layout 2

Effect on output:
Subsequent text characters are “emphasized” on the terminal as appropriate to the device type used (see table on page 1013).

Emphasized layout is reset by:
– the logical record control characters (NL, NP, HPA, VPA)
– the display control characters (EM1, EM3, EM4, DAR, DIS, NOR)
– the field control characters (EPA, SPA, NUM, CHS, COL, FLD)

&P.EM3
Emphasized layout 3

Effect on output:
Subsequent text characters are “emphasized” on the terminal (see table on page 1013).

Emphasized layout is reset by:
– the logical record control characters (NL, NP, HPA, VPA)
– the display control characters (EM1, EM2, EM4, DAR, DIS, NOR)
– the field control characters (EPA, SPA, NUM, CHS, COL, FLD)

&P.EM4
Emphasized layout 4

Effect on output:
Subsequent text characters are “emphasized” on the terminal (see table on page 1013).

Emphasized layout is reset by:
– the logical record control characters (NL, NP, HPA, VPA)
– the display control characters (EM1, EM2, EM3, DAR, DIS, NOR)
– the field control characters (EPA, SPA, NUM, CHS, COL, FLD)

&P.NOR
Normal layout

Effect on output:
Subsequent text characters are displayed normally on the terminal (i.e. emphasized layout is reset).

&P.SO
Shift out into character set extension in accordance with the table on page 1013.

Effect on output on certain terminals only (see table on page 1013):
Subsequent text characters are displayed on the terminal in the device-specific character set according to the table on page 1013.

Meaning for input:
Subsequent text characters were entered using the terminal-specific character set.

&P.SI
Shift into basic character set in accordance with the table on page 1013.

Effect on output:
Subsequent text characters are displayed on the terminal in the basic character set.

Meaning for input:
Subsequent text characters belong to the basic character set.

&P.DAR
Dark layout

Support for this operand is continued only for the sake of compatibility. Instead of &P.DAR, you should use the operands &P.EXT DIS and &P.EXT FLD.

Depending on the value of the DARPRINTABLE parameter, DAR is either a field or a logical display control character.

Effect on output:
The value of DARPRINTABLE is N' (default).
In the case of field-oriented display and structured output, DAR generates a start of field. Subsequent text characters are blanked and cannot be printed. The field is omitted from hardcopy.
DAR is reset by the logical record control characters (NL, NP, HPA, VPA) or the field control characters (EPA, NUM, CHS, COL and FLD).

The value of DARPRINTABLE is 'Y'.
Subsequent text characters are blanked. No start of field is generated.

DAR is reset by:
- the logical record control characters (NL, NP, HPA, VPA)
- the display control characters (EM1, EM2, EM4, NOR)
- the field control characters (EPA, SPA, NUM, CHS, COL, FLD)

&P.EXT DIS x
(data display terminals only) Set display attributes

Effect on output:
Subsequent text characters are highlighted on the terminal as appropriate to the device type used. Inverse video mode is ignored on 8110, 815x, 816x, 974x, 975x Data Display Terminals and 3270 terminals. On the 9763 Color Terminal, the attributes flashing, underscore, low intensity and combinations thereof are mapped to colors, depending on SIDATA.

x is a hexadecimal value that can be selected by means of the following equates:

&P.FL	flashing
&P.UND	underscore / italic (see EM2)
&P.BLK	invisible
&P.RIN	reduced intensity
&P.INV	inverse video
&P.RS	the attributes FL, UND, BLK, RIN and INV are reset. Note that RS does not have the same effect as NOR. NOR automatically selects the reduced intensity attribute.

x is either the value of one of these equates or the sum of a combination of equates.

e.g. `&P.EXT DIS &P.UND+&P.FL text` subsequent text is underscored and flashing

Any attribute not supported by the display terminal is ignored.

The highlight is reset by:
- the logical record control characters (NL, NP, HPA, VPA)
- the display control characters (EM1, EM2, EM3, EM4, DAR, DIS, NOR)
- the field control characters (EPA, SPA, NUM, CHS, COL, FLD)

3. Logical field control characters

&P.SPA
Start protected area

Effect on output (data display terminals only):
In the case of field-oriented display and structured output, SPA generates a start of field. Subsequent text characters are displayed at reduced intensity on the monitor of the data display terminal and are protected, i.e. they cannot be overwritten and returned to the data processing unit.
SPA is reset by the logical record control characters (NL, NP, HPA, VPA) or the field control characters (EPA, NUM, CHS, COL and FLD).

Note

> This function has a significant impact on the display attributes of certain terminals (newline on 8152 DDT, hardcopy function on 816x, 975x and 976x terminals). It should therefore be used with extreme caution.

&P.EPA
End protected area

Effect on output:
In the case of field-oriented display and structured output, EPA generates a start of field. Subsequent text characters are output to the terminal unprotected and displayed at high intensity.
EPA is reset by the logical record control characters (NL, NP, HPA, VPA) or by the field control characters (EPA, NUM, CHS, COL and FLD).

&P.NUM
Numeric area

Effect on output:
In the case of field-oriented display and structured output, NUM generates a start of field. Subsequent text characters are output to the terminal unprotected and displayed at high intensity.
You can enter only numeric data (digits, . * / + -) in this field.
NUM is reset by the logical record control characters (NL, NP, HPA, VPA) or by the field control characters (EPA, SPA, CHS, COL and FLD).

&P.CHS dd
Loadable character set
CHS is effective only on terminals of type 9763.

Effect on output:
In the case of field-oriented display and structured output, CHS generates a start of field and a loadable character set from the repertory available to the data terminal is selected for this field. dd is a two-digit decimal in the range 00-07 and designates the desired loadable character set. You can use the **TSTAT** macro to query the loadable characters sets available.

Example
If dd is 00, you address loadable character set 0 which has the symbolic name STACS0T in **DCSTA** (see MONCS).

Only characters from the selected character set can be entered in the generated field.

CHS is reset by:
– the logical record control characters (NL, NP, HPA, VPA)
– the display control characters (EM1, EM2, EM3, EM4, DAR, NOR, DIS)
– the field control characters (EPA, SPA, NUM, COL, FLD)

Note
> The user must ensure that the character sets of the data display terminal that are to be addressed logically via the control character CHS are loaded with the correct character sets.
> CHS is not supported in the system line.
> The character sets can be loaded physically in physical mode or with EXT TRA in line mode. Character sets can be created with the software product ICE.

&P.COL dd
Choice of colors
COL is effective only on color terminals of type 9763.

Effect on output:
In the case of field-oriented display and structured output, COL generates a start of field (even on monochrome screens). If the terminal is a 9763 color DDT, COL can be used to select a color from those available for this field. dd is a two-digit decimal in the range 00-07 designating the desired color.

00	default	01	blau	02	rot
03	magenta	04	grün	05	cyan
06	gelb	07	weiß		

Note that COL is not supported in the system line.

COL (color selected) is reset by:

- the logical record control characters (NL, NP, HPA, VPA)
- the display control characters (EM1, EM2, EM3, EM4, DAR, NOR, DIS)
- the field control characters (EPA, SPA, NUM, CHS, FLD)

&P.EXT DIM ll ccc
Physical screen dimension
EXT DIM is effective only on terminals of type 9763, and only if specified immediately after NP.

Effect on output:
EXT DIM can be used to select the screen format on display terminals of type 9763. The standard format is 24 x 80. Additional formats are 32 x 80, 43 x 80 and 27 x 132. The possible screen formats can be queried with **TSTAT**.
The number of lines ll is specified in two bytes. The number of columns ccc is specified in 3 bytes.
When the device is switched on, and during connection setup, the standard (default) format is reset automatically. The screen format is reset to the standard format by NP, at end of program, and when the screen is cleared by the system.
If a program is interrupted (K2, BKPT) in LINE mode, the screen format is not reset to the standard format.
CHS is not supported in the system line.

&P.EXT FLD x
(data display terminals only) Set field characteristics

Effect on output:
In the case of field-oriented display and structured output, a start of field is generated and certain characteristics are assigned to this new field. By default, this field is not printable. If emphasized display preceded this operand, it is set to reduced intensity for a protected field and to bright for all other fields.

x is a hexadecimal value selected by means of the following equates:

&P.PNS	protected field not sendable
&P.PRS	protected field sendable (automatic input)
&P.NUF	numeric field
&P.MOD	premodified field (only in extended line mode)
&P.MAR	markable field
&P.PRT	printable field
&P.INP	unprotected input field not numeric, not markable, not printable
&P.ASK	protected field with automatic tabulator (3270 terminals only)

x is either the value of one of these equates or the sum of a combination of equates.

e.g. `&P.EXT DIS &P.UND+&P.FL text` The field is numeric and markable

If a combination of field characteristics is defined, the output attribute 'reduced intensity' is retained. You must use the DIS control character to request other display attributes. Note that the field characteristics must always precede the output attributes.

FLD is reset by the logical record control characters (NL, NP, HPA, VPA) or by the field control characters (EPA, SPA, NUM, CHS, COL and FLD).

FLD is not supported in the system line.

Note that a markable field is of significance to the application only if the VTSUCB parameter READ=MODIFIED is used.

If the characteristics protected, not sendable and markable are assigned to a field, they correspond to the FHS attribute `PROTECTION=DETECTABLE` (protected selection field). If a field of this type is marked in read mode, the user receives only the field coordinates VPA and HPA and not the contents of the field. If you press the MAR key again, you can cancel 'marked' as an attribute of this field.

4. Logical local control characters

&P.LOC ctl
Set local attributes
LOC is effective only on data display terminals of type 9763.

Effect on input and output:
The attribute which is addressed with the control character ctl immediately following LOC is only created locally. ctl can be: EM1-EM4, NOR, DAR, DIS, CHS and COL.

A local attribute is valid for all subsequent characters up to the next start of field, if not explicitly reset before this. The attribute is reset using LOX (see page 1001). Local attributes are character-specific. When the character disappears, so also does the local attribute; any character that is inserted does not bear the local attribute, whilst the characters moved to the right retain their local attribute.

The following substitute representation is used on data display terminals other than type 9763 and for printers:
The local display attributes LOC EM1- LOC EM4, LOC NOR, LOC DIS and LOC DAR are replaced by the corresponding display attributes
EM1-EM4, NOR, DIS and DAR. There is no substitution for local character set attributes (LOC CHS) and color attributes (LOC COL).

In the case of LOC DAR, the subsequent text characters, which are not visible on the screen, can be printed. The value of DARPRINTABLE defines whether characters substituted for invisible text characters can be printed.

Local attributes can only be entered if the LOCIN=YES option is set in the VTSUCB. Otherwise they are deleted from the input message. Local attributes not supported logically by VTSU are always deleted from the input message.

Note

The logical display control characters EM1-EM4 and NOR, which are located after LOC and followed by EM1-EM4, NOR and DAR, are only effective once the local attribute has been reset (using LOX NOR or LOX LOX). These display control characters also have an effect on characters which are overwritten or inserted.

&P.LOX ctl
Reset local attributes

Effect on input and output:
Local attributes are reset with LOX.
ctl specifies which local attributes are reset.
ctl can be:

- NOR (local EM1, EM2, EM3, EM4, DIS, NOR or DAR is reset)
- CHS (local loadable character set is reset)
- COL (choice of color is reset)
- LOX (all local attributes are reset).

LOX resets the local attribute to the last valid non-local value.

The following substitute representation is used on data display terminals other than type 9763 and for printers:
LOX NOR and LOX LOX reset the attribute to the last display attribute which was not generated by a substitute layout. There is no substitute representation for local character set attributes (LOX CHS) and local color attributes (LOX COL).

Local attributes can only be received if the LOCIN=YES option is set in the VTSUCB. Otherwise they are deleted from the input message. Local attributes not logically supported by VTSU are always deleted from the input message.

Note

- In the case of EXTEND=NO, a logical display control character specified as the last character (EM1, EM2, EM3, EM4, DIS, DAR or NUM) acts on the subsequent input field. This provides a means of imposing the dark, numeric or emphasized attributes on a normal line mode input. If the output is followed not by an input but by another output, the control character has no effect.
- To specify a number of characteristics for a field, enter them in the following sequence: &P.CHS dd &P.COL dd &P.EXT &P.FLD xx &P.EXT &P.DIS xx. The logical display control characters must always follow the logical control characters for the field characteristics. Note that if you use the control character &P.COL, and the terminal supports color, the other display attributes are ignored. Physical sequences are still generated correctly. Same logical buffers are used for color and monochrome screen of type 9763.

- If the two logical control characters VPA and HPA are combined, the physical sequences generated differ depending on the positions of the control characters in the buffer.
  - If the control characters are at the start of the message, the cursor is positioned on the requested coordinates and a start of field with standard attributes is generated.
  - If the control characters are at the end of the message the cursor is positioned on the requested coordinates and special display formats of the data display terminal are reset to the standard display (normal, standard color, protected). The standard character set is retained.
  - If the control characters are within the message, the cursor is positioned on the requested coordinates and a start of field with standard attributes is generated. Special display formats of the data display terminal are reset to the standard display (normal, standard color, unprotected in line mode, protected in extended line mode). The standard character set is retained.
- Hardcopy support differs in line mode and extended line mode. In extended line mode, a hardcopy of the entire screen is produced. In line mode, a hardcopy is created only of the last field generated by a preceding control character.
- Fields generated with the EXT FLD control character must be defined as printable fields.

5. Logical control characters for printer support

Use of the logical control characters for printer support is only effective with EXTEND=NO.

&P.PLD
Partial line down (line subscripted)

Effect on output:
The printer moves half a line up and the subsequent text is printed half a line lower. This is reset by PLU, at the logical end of the line or at end of message.

&P.PLU
Partial line up (line superscripted)

Effect on output:
The printer moves half a line down and the subsequent text is printed half a line higher. This is reset by PLD, at the logical end of line or at end of message.

&P.VMI d
Vertical motion index (line spacing)
VMI is permitted only at the logical start of a page.

Effect on output:
Specifies the line spacing:

- d=1: normal spacing (1/6 inch).
- d=2: condensed spacing (1/8 inch).
- d=3: half-line spacing (1/12 inch).

Is reset by MLN at end of message. The number of lines per page is adjusted to match the new line spacing.

&P.HMI d
Horizontal motion index (character pitch)
HMI is permitted only at the logical start of a line. Any left margin that has been set with LM is reset.

Effect on output:
Specifies the character pitch:

- d=1: normal pitch (1/10 inch).
- d=2: condensed font A (character pitch 1/12 inch).
- d=3: condensed font B (character pitch 1/15 - 1/17 inch).

Reset by MLL and at end of message. The number of characters per line is automatically adjusted.

&P.NLQ
Near letter quality start

Effect on output:
On printers with near-letter quality capability, this feature is turned on by NLQ; otherwise NLQ is ignored. NLQ is reset at the end of the message.

&P.NLX
Near letter quality exit

Effect on output:
The output control character NLX turns off the near-letter quality feature.

&P.LM ddd
Left margin

Effect on output:
Sets a left margin. The first character of all subsequent lines is printed in the column specified by ddd.

LM is only permissible at the logical start of line and at the start of a message if there is no CAP at this point. It is reset by HMI, MLL and at the end of the message.
When the left margin is set, several successive backspaces (BS) are permitted. This enables the cursor to be positioned to the left of the left margin.

&P.PTS
Proportional type start

Effect on output:
A visually more pleasing printed result is achieved by individual character spacing.

PTS is effective until the end of the message or PTX.

Column counting is disabled and is not started again until an NL, NP, ASF, VPA or HPA following a PTX is received.

&P.PTX
Proportional type exit

Effect on output:
Deactivates proportional type.

&P.MLL ddd
Maximum line length

Effect on output:
ddd defines the new maximum number of characters per line. Also, the character spacing is set to 1/10 inch, and any left margin is reset. MLL only works at logical start of page and for printers. The defined maximum line length remains effective for the entire duration of the connection unless redefined.

&P.MLN ddd
Maximum line number

Effect on output:
ddd defines the new maximum number of lines per page. It also sets the line spacing to 1/6 inch. MLN only works at logical start of page and for printers. The defined maximum number of lines remains effective for the entire duration of the connection unless redefined.

&P.VPA ddd
Vertical position absolute (three-digit decimal)

Effect on output:
The output text starts in the line designated by ddd.

&P.HPA ddd
Horizontal position absolute (three-digit decimal)

Effect on output:
Subsequent characters in the current line are output starting at the column designated by ddd.

&P.ASF d
Automatic sheet feeding (sheet insertion / ejection control) d= decimal number

Effect on output:

- For d=0 at the end of the message a sheet ejection is triggered.
  When a form feed attachment is used: switchover to tractor.
- For d=1,2,3 the sheet ejection is combined with a sheet feed from the feed tray designated by d.
- For d=9 sheet insertion takes place from the form feed attachment (9013 Printer only).

&P.CAP
Continue actual position (no positioning to the start of the next line at the beginning of a message)

Effect on output:
Printing starts at the current position of the printer carriage. No resetting of logical control characters at end of message or start of message. Additionally, multiple PLUs and PLDs are allowed. However, only one PLU or PLD is reset at logical end of line.

To ensure that there is a defined initial state, the printer must be positioned to the start of a line with NL, NP or VPA when CAP is used for the first time within an output sequence. CAP is permitted only as the first character in a message. Overwriting may occur when CAP is used for the first time or after a reset sequence (e.g. HMI reset) if the cursor is not explicitly positioned on the start of a new line.

6. Logical control characters with special functions

&P.DEL
Delete

Effect on output:
The character is removed from the output text and is not transmitted to the terminal.

&P.BS
Backspace (only display terminals with APL capability, and printers)

Effect on output:
The next text character is superimposed on the preceding character (to form a compound character not contained in the character set).

A multiple BS is only allowed when a left margin has been set previously with LM.

Meaning for input:
The next text character and the preceding one are treated as a single unit.

Note

For the 9022 Printer, BS (backspace) following SO (shift out into character set extension) is ignored.

&P.SUB
Substitute

Effect on output:
This logical control character and all other EBCDIC control characters (code < X'40') which are not logical control characters are represented on the terminal by the valid substitute character (macro **TCHNG** SUB=OUTIN).

Effect on input:
The valid substitute character has been detected in the terminal input and replaced (only if requested via the macro **TCHNG** SUB=OUTIN, see the “TIAM” manual [16]).

&P.ESC
Escape

Effect on output:
Generates the control character ESC (EBCDIC code X'27'). This is transferred together with the next character in unchanged form. This permits the use, in line mode, of terminal device functions that are not logically supported (see descriptions of the respective terminals).

Note

This control character causes the column and line counts by VTSU to be suppressed (no overflow control).
Column counts are resumed by the control characters NL, NP or VPA, line counts (overflow control) are resumed by the control character NP or VPA.

&P.DC4
Treated like ESC by VTSU

Effect on output:
Generates the control character DC4 (EBCDIC code X'3C'). This is transferred together with the following character to the terminal. This permits the use, in line mode, of terminal device functions that are not logically supported (cf. descriptions of the respective terminals).

The effect on column and line counting is as described in the note on the ESC control character.

&P.HT
Horizontal tabulation

Effect on output:
Generates the control character HT (EBCDIC code X'05'), which is passed unchanged to the terminal (cf. descriptions of the respective terminals).

The effect on column and line counting is as described in the note on the ESC control character.

&P.VT
Vertical tabulation

Effect on output:
Generates the control character VT (EBCDIC code X'0B'), which is passed unchanged to the terminal (cf. descriptions of the respective terminals).

The effect on column and line counting is as described in the note on the ESC control character.

&P.EXT TRA d ll
Transparent output

Effect on output:
EXT TRA allows the user to transfer control character sequences unchanged (transparently) to data display terminals and printers.
With d, the user can select the devices to which the transparent character sequence is to be transferred:

d = X'00'
The control character sequence is transferred for all device types.

d = device type supplied in **TSTAT**:
The control character sequence is transferred only for the specified device type.

With ll, the user specifies the length of the transparent control character sequence. ll is specified in two bytes in either decimal (00-99) or binary form (X'0000'-X'7FFF'). The first five bytes (EXT TRA d ll) are not counted as part of the length specification. If the length is greater than the maximum device buffer, EXT is converted to SUB.

Note

> If you use the logical control character EXT TRA you can specify any physical control character.
> If you use EXT TRA in order to modify positioning, correct overflow control is no longer guaranteed.

&P.EXT RPT dd
The subsequent displayable or NULL character is repeated dd times (repeat symbol)

Effect on output:
EXT RPT enables you to reduce the size of your buffer in the case of multiple repeats of a character.
dd is the number of times you want the subsequent character repeated. dd is specified in two bytes, either in decimal (00-99) or in binary notation (X'0000'-X'7FFF').

Notes on the logical control characters

a) If a control character is illegal for a particular output device or in its current position, it is ignored together with any associated number following it, or it is replaced (see description of the individual control characters).

b) If a logical control character that expects a number to follow it is in fact followed by bytes that do not form a legal number (no number or an impossible number), the control character is replaced by SUB and the subsequent characters are treated as text.

c) If the number following the logical control character is only illegal in special cases (e.g. too large for the current line length), it is processed according to the procedure described under a).

d) The control characters ASF, VPA and HPA have the effect of a logical end of line and act on the preceding control characters in the same way as NL (reset character set 2 etc.).

Effect of logical display control characters on data display terminals

Terminal	Effect of logical display control characters					
	NOR	EM1	EM2	EM3	EM4	DAR
8110 TTY [1]	- [2]	-	-	-	-	-
8150	-	-	-	-	-	
8151	still	flashing	flashing	flashing	flashing	-
8152	roman	italics	italics	italics	italics	-
8160 8162 } OM 1 [3]	low int. roman still	low int. roman flashing	low int. italics still	high int. italics still	high int. italics still	blanked
8160 OM 2	high int. roman still	high int. roman flashing	high int. italics still	low int. roman still	low int. italics still	blanked
9748 9749 9750 9751 } OM 1	low int. roman still	low int. roman flashing	low int. under-scored still	high int. roman still	high int. under-scored still	blanked
9748 9749 9750 9751 } OM 2	high int. roman still	high int. roman flashing	high int. under-scored still	low int. roman still	low int. under-scored still	blanked

Terminal	**Effect of logical display control characters**					
	NOR	**EM1**	**EM2**	**EM3**	**EM4**	**DAR**
9752	yellow roman	yellow flashing	white roman	green roman	red roman	blanked
9755 9756 OM 1 9758	low int. roman	low int. roman flashing	low int. underscored	high int. roman	high int. underscored	blanked
9755 9756 OM 2 9758	high int. roman	high int. roman flashing	high int. underscored	low int. roman	low int. underscored	blanked

Terminal	**Effect of logical display control characters**					
	NOR	**EM1**	**EM2**	**EM3**	**EM4**	**DAR**
9763 and 9759 OM 1 monochrome screen	low int. roman	low int. roman flashing	low int. under-scored	high int. roman	high int. under-scored	blanked
9763 and 9759 OM 2 monochrome screen	high int. roman	high int. roman flashing	high int. under-scored	low int. roman	low int. under-scored	blanked
9763 OM 1 color screen	yellow	cyan	white	green	red	blanked
9763 OM 2 color screen	green	red	red	yellow	white	blanked
3270 OM 1	low int roman	high int. roman	high int. roman	high int. roman	high int. roman	blanked
3270 OM 2	high int. roman	low int. roman	low int. roman	low int. roman	low int. roman	blanked
3279 OM 1	green	red	red	red	red	blue
3279 OM 2	green	white	white	white	white	green
8121 8122	normal	italics	italics	italics	italics	-
9001	normal	under-scored	under-scored	under-scored	under-scored	-
9002	normal	italics	under-scored	italics	italics under-scored	-
9003	normal	italics	red	italics	red and italics	-
9004	normal	shadow script	under-scored	fett	bold and under-scored	-
9013	normal	under-scored	under-scored	bold	bold and under-scored	-
9012	normal	under-scored	under-scored	bold	bold and under-scored	-
9011-18/19	normal	italics [4]	under-scored	bold	bold and under-scored	-

Terminal	Effect of logical display control characters					
	NOR	EM1	EM2	EM3	EM4	DAR
9001-31 / 8931	normal	italics [4]	underscored	bold	bold and underscored	-
9021	normal	italics	underscored	bold	bold and underscored	-
9022	normal	shadow script	underscored	bold	bold and underscored	-

[1] TTY: PT80, T100, T1000

[2] -: control character is ignored

[3] OM: operating mode

[4] Only for connection to a 9763 DDT or a BAM controller, otherwise underscored or no effect

Effect of logical display control characters on data display terminals

Terminal	Effect of logical display control characters				
	SO	SI	SPA	EPA	NUM
8110 TTY [1]	- [2]	-	-	-	-
8150	-	-	protected low int.	unprotected high int.	-
8151	-	-	protected low int.	unprotected high int.	-
8152	APL character set	first character set	protected low int.	unprotected high int.	-
8160	-	-	protected low int.	unprotected high int.	unprotected high int. [3]
8162	second character set	first character set	protected low int.	unprotected high int.	unprotected high int.[3]
9748 9749 9750/9751	-	-	protected low int.	unprotected high int.	unprotected high int.[3]
9752	-	-	protected yellow	unprotected green	unprotected green [3]
9755 9756 9758	-	-	protected low int.	unprotected high int.	unprotected high int.[3]
9763 / 9759 monochrome screen	-	-	protected low int.	unprotected high int.	unprotected high int.[3]
9763 color screen	-	-	protected yellow	unprotected green	unprotected green [3]
3270	-	-	protected low int.	unprotected high int.	unprotected high int.[3]
3279 OM 1 [4]	-	-	white	red	red
3279 OM 2	-	-	white	white	white
812x	-		-	-	-
9001	-	-	-	-	-
9002 [5]	second character set	first character set	-	-	-
9003	second character set	first character set	-	-	-

Terminal	Effect of logical display control characters				
	SO	SI	SPA	EPA	NUM
9004	second character set	first character set	-	-	-
9013 [5]	character set extension [6]	basic character set [7]	-	-	-
9012	character set extension [6]	basic character set [7]	-	-	-
9011-18/19	second character set	first character set	-	-	-
9001-31/ 8931	-	-	-	-	-
9021	secondary font	primary font	-	-	-
9022	character set extension [8]	basic character set [9]	-	-	-

[1] TTY: PT80, T100, T1000

[2] -: control character is ignored

[3] entries must be numeric

[4] OM: operating mode

[5] depending on printer type (see relevant printer manual)

[6] right half of ISO 8-bit code table; practical for e.g. teletex character set

[7] left half of ISO 8-bit code table

[8] right half of ISO 8-bit code table; practical for e.g. teletex character set

[9] left half of ISO 8-bit code table

Applicability of the logical control characters to individual printers and data display terminals

Type	PLD PLU	LM	PTS PTX	VPA	HPA	ASF	MLL	MLN	BS	CAP	ESC DC4	HT	VT
9004	X	X	X	X	X	X	X	X	X	X	X	X	X
9001	X	-	-	-	-	-	X	X	-	X	X	X	X
9003	-	-	-	-	-	-	X	X	X	X	X	X	X
9002	-	-	-	X	X	-	X	X	X	X	X	X	X
9013	X	X	X	X	X	X	X	X	X	X	X	X	X
9012	X	X	X	X	X	X	X	X	X	X	X	X	X
9011 18/19	X	-	X	-	X	X	X	X	X	X	X	X	X
9001-31/ 8931	X	-	X	-	X	-	X	X	X	X	X	X	X
9022	X	X	X	X	X	X	X	X	X	X	X	X	X
9021	X	-	X	X	X	X	X	X	X	X	X	X	X
812x	-	-	-	-	-	-	X	X	X	X	X	-	-
816x 975x 974x	-	-	-	X	*	-	-	-	-	-	X	X	X
976x	-	-	-	X	*	-	-	-	-	-	X	X	X
3270	-	-	-	X	*	-	-	-	-	-	X	X	X
3279	-	-	-	X	*	-	-	-	-	-	X	X	X

Meaning:

X	function is initiated
-	logical control character is suppressed
*	if preceded by a legal VPA, the logical control character is suppressed

Note

– The 9022 Printer ignores proportional type unless the current font supports proportional spacing.

– Proportional type is impractical with the 9013 Printer unless a suitable character set is selected.

– When used on the 9013 and 9002 Printers, the control character VPAddd causes the execution of a number of line feeds defined by ddd.

– When used on the 9002 Printer, the control character HPAddd either adds the number of blanks defined by ddd or defines the column as of which the subsequent characters are to be output. You can select either of these two options. The insertion of blanks is the default.

Applicability of logical control characters for output

Terminal	CHS	LOC LOX	EXT DIM	EXT TRA	NLQ NLX	EXT DIS	EXT FLD	COL	EXT RPT
9763	X	X	X	X	-	X	X	X	X
975x 9748 9749 816x 3270	-	[1]	-	X	-	X	X	-	X
9001 9002 9003 9004 9013	-	[1]	-	X	-	-	-	-	X
9012	-	[1]	-	X	-	-	-	-	X
9011-18/19	-	[1]	-	X	X	-	-	-	X
9001-31/ 8931	-	[1]	-	X	X	-	-	-	X
9022	-	[1]	-	X	-	-	-	-	X
9021	-	[1]	-	X	-	-	-	-	X

[1] substitution takes place (see LOC and LOX control characters)

Meaning:

X function is initiated
- logical control character is suppressed

Note

On 9763 Data Display Terminals, only the screen formats supplied with **TSTAT** are supported for the logical control character EXT DIM.

Effect of output attributes

Terminal	FL	UND	BLK	RIN	INV
8110	- [1]	-	-	-	-
815x	-	-	-	-	-
8160	flashing	italics	blanked	low intensity	-
9750	flashing	underscored / inverse [2]	blanked	low intensity	-
9755	flashing	underscored / inverse [3]	blanked	low intensity	-
9758 9756	flashing	underscored-inverse [3]	blanked	low intensity	inverse
9763 9759	flashing	underscored-inverse [3]	blanked	low intensity	inverse
3270	-	-	blanked and not printable	low intensity	-

[1] -: control character is ignored

[2] set by jumper or ROM

[3] selectable by SIDATA

Effect of field attributes

Terminal	PNS	PRS	NUF	MOD	MAR	PRT	ASK
8110	- [1]	-	-	-	-	-	-
815x	-	-	-	-	-	-	-
8160	protected not sendable	protected sendable	numeric	pre-modified	markable	printable	-
9750	protected not sendable	protected sendable	numeric	pre-modified	markable	printable	-
9755	protected not sendable	protected sendable	numeric	pre-modified	markable	printable	-
9758 9756	protected not sendable	protected sendable	numeric	pre-modified	markable	printable	-
9763 9759	protected not sendable	protected sendable	numeric	pre-modified	markable	printable	-
3270	protected not sendable	-	numeric	pre-modified	selectable [2]	printable [3]	protected automat. tabulator

[1] -: control character is ignored

[2] the first character in the field is a destination character

[3] on a 3270 terminal, a non-printable field is automatically blanked

Character spacing and line spacing for the individual printers (in inches)

Printer	HMI1	HMI2	HMI3	VMI1	VMI2	VMI3
9004	1/10	1/12	1/15	1/6	1/8	1/12
9001	1/10	1/12	1/17	1/6	1/8	1/12
9003	1/10	1/12	1/15	1/6	1/8	1/12
9002	1/10	1/10	1/10	1/6	1/6	1/6
9013	1/10	1/12	1/15 1st ch.set only	1/6	1/8	1/12
9012	1/10	1/12	1/12	1/6	1/8	1/12
9011 18/19	1/10	1/12	1/15	1/6	1/8	1/12
900131/8931	1/10	1/12	1/15	1/6	1/8	1/12
9022	1/10	1/12	1/15	1/6	1/8	1/12
9021	1/10	1/12	1/15	1/6	1/8	1/12
812x	1/10	1/10	1/10	1/6	1/6	1/6

Note

- On the 9011-31/8931 Printer, bold (EM3, EM4) is not executed after HM13.
- On the 9021 Printer, HM13 is executed with 1/15“ only when a suitable font has been selected; otherwise the next smallest character spacing is used.

Effect of reset at connect time on logically supported functions

Printer	HMI	VMI	PTS	SO	CHS	ASF	EM1-4	NLQ	CR
9001	1/10	S	-	-	ZV1	-	R	-	S
9004	S	S	R	R	-	ASF1	R	-	-
9013	S	S	R	R	S	ASFn, n(S)	R	-	S
9012	M	M	M	R	M	ASFn, n(M)	R	M	M
9011 18/19	M	M	M	R	M	Tract ASFn, n(M)	EM1: M EM2: R EM3: M	M	M
9001-31/8931	S	S	R	-	ZV1	-	R	S	-
9022	S	S	R	R	R	ASF1	R	-	-
9021	S	S	R	R	R	ASF1	R	-	-
VTSU default	**1/10**	**1/6**	**R**	**R** [1]	**CS1**	**-**	**R**	**R**	**CR**

[1]) Left half of the ISO 8-bit code table for printers with 8-bit character sets; otherwise basic character set

Meaning:

S	Reset to switch setting
M	Reset to menu setting
R	Reset
-	No logical support for function
CS	Character set
n(S)	n defined by switch setting
n(M)	n defined by menu selection

Note

– VTSU default
It is assumed for the logical support of these functions that the VTSU default values are set by menu or switch settings. Any differences from the default settings are the responsibility of the user.

– Automatic sheet feed
For the 9011-18/19 Printer, the menu setting for automatic sheet feed (ASF) simply means selection of tractor or sheet feeder.

Layout of the DSECT

```
           VTCSET BSP
1 *
1 *              VIRTUAL TERMINAL CONTROL CHARACTER SET
1 *
1 *
1 *              LOGICAL RECORD DELIMITERS
1 *
1 BSPNL    EQU   X'15'           LOGICAL LINE END (CONT NEXT LINE)
1 BSPNP    EQU   X'0C'           LOGICAL PAGE END (CONT NEXT PAGE)
1 BSPCL    EQU   X'0D'           LOGICAL LINE END (CONT SAME LINE)
1 BSPVPA   EQU   X'29'           LOG VERTICAL POS ABSOLUT (CONT LINE N)
1 BSPHPA   EQU   X'2A'           LOG HORIZONT POS ABSOLUT (CONT COL N)
1 BSPASF   EQU   X'21'           LOG SHEED FEDDING FROM CASETTE N  D1
1 BSPCAP   EQU   X'20'           CONTINUE ACTUAL POSITION AT MSG BEGIN
1 *
1 *              LOGICAL UNIT DELIMITERS
1 *
1 BSPEM1   EQU   X'1D'           EMPHASIZED LAYOUT 1
1 BSPEM2   EQU   X'1F'           EMPHASIZED LAYOUT 2
1 BSPEM3   EQU   X'13'           EMPHASIZED LAYOUT 3
1 BSPEM4   EQU   X'14'           EMPHASIZED LAYOUT 4
1 BSPNOR   EQU   X'1E'           NORMAL LAYOUT
1 BSPDAR   EQU   X'12'           DARK LAYOUT
1 BSPPLD   EQU   X'2B'           PARTIAL LINE DOWN
1 BSPPLU   EQU   X'2C'           PARTIAL LINE UP
1 *
1 BSPSO    EQU   X'0E'           SHIFT OUT TO 2ND CHARACTER SET
1 BSPSI    EQU   X'0F'           SHIFT IN TO NORMAL CHARACTER SET
1 *
1 BSPSPA   EQU   X'36'           START PROTECTED AREA
1 BSPEPA   EQU   X'08'           END PROTECTED AREA
1 BSPNUM   EQU   X'11'           START NUMERIC (UNPROTECTED) AREA
1 *
1 BSPCHS   EQU   X'06'           CHARACTER SET D1D2
1 BSPCOL   EQU   X'17'           COLOUR CHOICE
1 BSPLOC   EQU   X'09'           LOCAL ATTRIBUTE START S1
1 BSPLOX   EQU   X'0A'           LOCAL ATTRIBUTE EXIT S1
1 *
1 BSPVMI   EQU   X'24'           VERTICAL MOVEMENT INDICATOR  D1
1 BSPHMI   EQU   X'23'           HORIZONTAL MOVEMENT INDICATOR  D1
1 BSPLM    EQU   X'38'           LEFT MARGIN  D1D2D3
1 BSPPTS   EQU   X'1A'           PROPORTIONAL TYPING START
1 BSPPTX   EQU   X'1B'           PROPORTIONAL TYPING END
1 BSPMLL   EQU   X'33'           MAXIMAL LINE LENGTH
1 BSPMLN   EQU   X'35'           MAXIMAL LINE NUMBER (ON PAGE)
1 BSPNLQ   EQU   X'39'           NEAR LETTER QUALITY START
```

```
1 BSPNLX   EQU   X'3B'            NEAR LETTER QUALITY EXIT
1 *
1 *              SPECIAL FUNCTIONS
1 *
1 BSPDEL   EQU   X'07'            DELETE
1 BSPBS    EQU   X'16'            BACKSPACE
1 BSPSUB   EQU   X'3F'            SUBSTITUTE
1 *
1 *              DELIMITER EXTENSION
1 *
1 BSPEXT   EQU   X'3E'            DELIMITER EXTENSION BYTE
1 *
1 *              EXTENDED LOGICAL DELIMITERS
1 *
1 BSPTRA   EQU   C'T'             TRANSPARENT OUTPUT X1L1L2
1 BSPDIM   EQU   C'D'             DIMENSION OF SCREEN D1D2D3D4D5
1 BSPRPT   EQU   C'R'             REPEAT NEXT CHARACTER NN TIMES
1 BSPDIS   EQU   C'I'             SET DISPLAY ATTRIBUTES
1 BSPRS    EQU   X'00'             RESET DISPLAY ATTRIBUTES
1 BSPFL    EQU   X'01'             FLASHING
1 BSPUND   EQU   X'02'             UNDERSCORED
1 BSPBLK   EQU   X'04'             BLANKED
1 BSPRIN   EQU   X'08'             REDUCED INTENSITY
1 BSPINV   EQU   X'10'             INVERSE
1 BSPFLD   EQU   C'F'             SET FIELD CHARACTERISTICS
1 BSPINP   EQU   X'00'             INPUT FIELD
1 BSPPNS   EQU   X'01'             PROTECTED NOT SENDABLE
1 BSPPRS   EQU   X'20'             PROTECTED SENDABLE
1 BSPNUF   EQU   X'02'             NUMERIC
1 BSPMOD   EQU   X'04'             PRE-MODIFIED
1 BSPMAR   EQU   X'08'             MARKABLE
1 BSPPRT   EQU   X'10'             PRINTABLE
1 BSPASK   EQU   X'40'             AUTOMATIC SKIP
1 *
1 *
1 *              PHYSICAL UNIT DELIMITERS
1 *
1 BSPESC   EQU   X'27'            ESCAPE  X
1 BSPDC4   EQU   X'3C'            DC4  X
1 BSPHT    EQU   X'05'            HORIZONTAL TABULATION
1 BSPVT    EQU   X'0B'            VERTICAL TABULATION
1 *
1                *,VTCSET     080    941024   53531028
```

An **example** of extended line mode is given at the end of the **WRTRD** macro description (Example 5).

VTSUCB – Create VTSU parameters for input/output

General

Application areas:	Input/output of files and records; see page 156 Data terminal communication; see page 160
Macro type:	Type S, MF format **3**: C/D/L/M form; see page 29

- This description applies to VTSU V13.3A

Macro description

The VTSU control block (VTSUCB) is a program interface that allows the user to create VTSU parameters for input and output with the **RDATA**, **WROUT** and **WRTRD** macros. The following macros can be used to access the input/output interface:

WROUT record,error[,PARMOD=31],VTSUCBA=addrvtsucb

or WRTRD record1,,record2,,[length],error,[,PARMOD=31],VTSUCBA=addrvtsucb

or RDATA record,error,[length][,A][,PARMOD=31],VTSUCBA=addrvtsucb

where the VTSUCBA operand specifies an address (addrvtsucb) at which the VTSU parameters start. These specifications replace the edit parameters of these macros. The **VTSUCB** macro creates such a parameter list.

The **RDATA**/**WROUT**/**WRTRD** interfaces with edit parameters can be used parallel to the **VTSUCB** interfaces. New functions (as of VTSU V9.0B), however, can only be used via **VTSUCB**, since the edit parameters are no longer being extended.

Extended character sets are supported by the CCSNAME and CODETR operands. These operands are evaluated only if MODE=LINE/EXTEND/INFO/FORM/PHYS is specified.

Macro format and description of operands

```
VTSUCB

[{
   MODE=LINE,BELL={ NO  },GETFC={ NO  },HCOPY={ NO  },NOPOS={ NO  }
   ---------      { --  }       { --  }       { --  }       { --  }
                  { YES }       { YES }       { YES }       { YES }

             ,LOW={ NO  },NOLOG={ NO  }[,RETINF={ *NONE }],HOM={ NO  }
                  { --  }       { --  }         { ----- }      { --  }
                  { YES }       { YES }         { xx    }      { YES }

             [,CCSNAME={ *EXTEND }],SPECIN={ N }
                       { ccsname }         { - }
                                           { I }
                                           { C }

             ,ENCOUT={ NO  },ENCIN={ NO  },INFOLR={ NO  }
                     { --  }       { --  }        { --  }
                     { YES }       { YES }        { YES }

   MODE=EXTEND ,BELL={ NO  },GETFC={ NO  },LOW={ NO  }[,CCSNAME={ *EXTEND }]
                     { --  }       { --  }     { --  }          { ccsname }
                     { YES }       { YES }     { YES }

             ,LOCIN={ NO  },UPDATE={ NO  },CURPOS={ NO  }
                    { --  }        { --  }        { --  }
                    { YES }        { YES }        { YES }

             ,AUTOTAB={ STD }],HCOPY={ NO  },READ={ UNPROT   }
                      { --- }        { --  }      { ------   }
                      { YES }        { YES }      { MODIFIED }
                      { NO  }

             ,ENCOUT={ NO  },ENCIN={ NO  },INFOLR={ NO  }
                     { --  }       { --  }        { --  }
                     { YES }       { YES }        { YES }
}]

:                                                                    :
```

VTSUCB (cont.)

:

$$\left[\left\{\begin{array}{l}
\text{MODE=(MIXED,inmod,outmod) ,LOW=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,HOM=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,COPY=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\\
\qquad\qquad\text{,BELL=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,GETFC=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,UPDATE=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,NOPOS=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\\
\qquad\qquad\text{,READ=}\left\{\begin{array}{c}\underline{\text{UNPROT}}\\\text{MODIFIED}\end{array}\right\}\text{,NOLOG=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,IHDR=}\left\{\begin{array}{c}\underline{\text{YES}}\\\text{NO}\end{array}\right\}\\
\qquad\qquad\text{,LOCIN=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,CODETR=}\left\{\begin{array}{c}\underline{\text{YES}}\\\text{NO}\end{array}\right\}\text{,CURPOS=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\\
\qquad\qquad\text{,OHDR=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{ [,CCSNAME=}\left\{\begin{array}{c}\text{*EXTEND}\\\text{ccsname}\end{array}\right\}\text{] ,RETINF=}\left\{\begin{array}{c}\underline{\text{*NONE}}\\\text{xx}\end{array}\right\}\\
\qquad\qquad\text{,AUTOTAB=}\left\{\begin{array}{c}\underline{\text{STD}}\\\text{YES}\\\text{NO}\end{array}\right\}\text{,SPECIN=}\left\{\begin{array}{c}\underline{\text{N}}\\\text{I}\\\text{C}\end{array}\right\}\\
\qquad\qquad\text{,ENCOUT=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,ENCIN=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,INFOLR=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\\
\text{MODE=INFO ,BELL=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,GETFC=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,LOW=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{[,CCSNAME=}\left\{\begin{array}{c}\text{*EXTEND}\\\text{ccsname}\end{array}\right\}\text{]}\\
\qquad\qquad\text{,NOLOG=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,ENCOUT=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,ENCIN=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\\
\text{MODE=PHYS ,IHDR=}\left\{\begin{array}{c}\underline{\text{YES}}\\\text{NO}\end{array}\right\}\text{,LOW=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,OHDR=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{[,CCSNAME=}\left\{\begin{array}{c}\text{*EXTEND}\\\text{ccsname}\end{array}\right\}\text{]}\\
\qquad\qquad\text{,CODETR=}\left\{\begin{array}{c}\underline{\text{YES}}\\\text{NO}\end{array}\right\}\text{,ENCOUT=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,ENCIN=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}\text{,INFOLR=}\left\{\begin{array}{c}\underline{\text{NO}}\\\text{YES}\end{array}\right\}
\end{array}\right\}\right]$$

:

```
VTSUCB (cont.)

:                                                                        :

  ⎧ MODE=FORM ,LOW=⎧ NO  ⎫[,CCSNAME=⎧ *EXTEND ⎫]
  ⎪                ⎩ YES ⎭          ⎩ ccsname ⎭
  ⎪
  ⎪            ,ENCOUT=⎧ NO  ⎫,ENCIN=⎧ NO  ⎫,INFOLR=⎧ NO  ⎫
  ⎪                    ⎩ YES ⎭       ⎩ YES ⎭        ⎩ YES ⎭
 [⎨                                                              ⎬]
  ⎪ MODE=TRANS ,ENCOUT=⎧ NO  ⎫,ENCIN=⎧ NO  ⎫,INFOLR=⎧ NO  ⎫
  ⎪                    ⎩ YES ⎭       ⎩ YES ⎭        ⎩ YES ⎭
  ⎪
  ⎪ MODE=CHIP ,ENCOUT=⎧ NO  ⎫,ENCIN=⎧ NO  ⎫
  ⎩                   ⎩ YES ⎭       ⎩ YES ⎭

,MF=L / C / D / M

[,PARAM=addr / (r)]

,PREFIX=Y / p

,MACID=VTC / macid
```

MODE=

LINE

The current terminal is to be treated as a line or page terminal. The message can be structured with logical control characters (see the **VTCSET** macro).
Any other control characters are invalid for output and are converted into a user-defined substitute character (see the MODIFY-TERMINAL-OPTIONS SUBSTITUTE-CHARACTER=command).
If SYSOUT is not a terminal, only the logical control characters NL and NP are evaluated, e.g. for output to printers in batch mode.
The device-specific message header is not supplied during input.

EXTEND

(For 9749, 975x, 9763, 816x and 3270 Data Display Terminals only, EXTEND is processed internally as line mode in the case of printers)

The current terminal should be treated as a logical line or page terminal. Text output is protected and at reduced brightness unless specified otherwise. The message can be structured by logical control characters (see **VTCSET** macro). Keys RU, EFZ, AFZ and LSP are locked.

In the case of TIAM applications, areas which the user can use for input start with EPA, DAR or NUM and end with SPA. With 3270 terminals, note that logical control characters occupy space on the screen. However, a series of logical control characters requires only one space.
In TIAM applications NIL is treated as a permissible character on input and output, and is sent to the terminal by the program and vice versa. In the case of 3270 data display terminals, note that NIL characters are not transferred to the DVA. VTSU-B fills fields which are returned truncated on input with NIL characters back to their original length. This ensures that the fields are always returned to the user in their original (output) length.
In TIAM applications the start of an output message is mapped at the start of the next line following the cursor. If the message does not begin with VPA, the screen is cleared, starting at the cursor and before the first text character.
If the end of the screen is reached during output, output is continued at the top of the screen. This continuation is always protected until the start of the next field. Screen overflow control is ineffective here.

If the NL control character is recognized in an input message, processing continues and the return code X'2C' is supplied at the **WRTRD** interface or the return code (main code) X'0018' at the **VTSUCB** interface.

INFO
Messages can be mapped to a special information line (system line) without destroying important data on the terminal.

The entry is intended particularly for user programs sending messages to terminals “asynchronously” without knowing the current terminal display.

Mapping is performed
- protected, in a hardware display line (e.g. 9749, 9750, 9752 Data Display Terminals), or
- protected, in the last line on the screen (e.g. 816x, 9751, 9753, 3270 Data Display Terminals) if specified in the user program (see the **TCHNG** macro, INFOLIN operand), after previous output with MODE=FORM or MODE=PHYS.
- in all other cases, as a normal line mode message.

If the message length exceeds one screen line, the message is split up and output line by line.
The system observes the waiting time specified in the MODIFY-TERMINAL-OPTIONS OVERFLOW-CONTROL=TIME( ) command.

The system line is reset automatically the first time an input is made following an output to the system line.

Input with MODE=INFO is treated as a line mode input.

Note

If, before the system line output, **WRTRD** is used in line mode, an input field is set up automatically.

(MIXED,inmode,outmode)
The application program uses a combination of different modes for input and output. Only MODE=LINE, EXTEND, FORM and PHYS may be combined. MODE=CHIP, INFO or TRANS may not be combined.
The default value of the LOW parameter, for input and output, depends on the defined input mode. If MODE=PHYS, then LOW=YES; if MODE=LINE, EXTEND or FORM, then LOW=NO. All other parameter values are defined in accordance with the required input/output modes.

PHYS
The messages are to be output to or read from the terminal physically, i.e. without editing by the system. This permits special device functions to be executed for which the LINE or FORM mode is insufficient.

FORM
Specifies format mode. The user program works with the “Terminal Mapping Support” software component (FHS), which edits the message in a form suitable for output to a particular terminal.

TRANS
Output data is to be transmitted in “transparent” form, i.e. consisting of arbitrary binary characters (5, 7 or 8 bits per character, depending on the device code) which are not converted during transmission. If the transmission path was not generated “potentially transparent”, output is rejected with return code X'04' at the **WROUT**/**WRTRD** interface.

CHIP
The output message is forwarded to the chipcard terminal using the device protocol (810 protocol). The message must be created in expanded mode (see the “Data Display Terminals” manuals). If the chipcard terminal is not addressable, output is rejected with return code X'81' or X'82' (subcode 2) at the **VTSUCB** interface.

During input, a check is made as to whether the message is being sent from a chipcard terminal, and the device protocol is removed. The function key code is placed in front of the input message as the first byte. Input messages that are not sent from the chipcard terminal are converted to short code K14.

Note

The MODE=CHIP operand is not permitted in the **RDATA** macro.

AUTOTAB=
The automatic tab jump from one unprotected field to the next is set. In mixed mode this parameter is accepted only if input and output mode have the value EXTEND. Otherwise it is ignored.

STD
Processing depends on the EXPROPOS operational parameter.

YES
As soon as you enter a character at the end of an unprotected field, the cursor automatically jumps forward from this unprotected field to the next unprotected field (including when EXPROPOS=Y).

Notes

- With 3270 terminals it is always possible to move the cursor with the arrow keys in protected fields. If however AUTOTAB=YES, the cursor automatically jumps from one input field to the next as soon as a character is entered at the end of an input field.
- The EXPROPOS operational parameter is ignored by the 3270 terminal.

NO
The cursor does not jump automatically (including when EXPROPOS=N).

BELL=
Determines whether an audible signal is provided on output.

NO
No audible signal is provided for output.

YES
An audible signal is heard on output at the end of the message (applies only to 9749, 975x, 9763, 816x and 3270 Data Display Terminals with special hardware feature).

CCSNAME=
Defines the name of the character set to be used for this message. The code name of the EBCDIC variant must be specified. The name of the corresponding ISO code variant is rejected automatically. The name may be up to 8 bytes long. If the character set used is changed prior to the new output, the screen is cleared automatically.

ccsname
Name of any EBCDIC code. If no name is specified, standard mode is assumed automatically. This is either a 7-bit mode or an 8-bit mode activated by means of the MODIFY-TERMINAL-OPTIONS command.

***EXTEND**
The extended user default code is used automatically.

CODETR=
Specifies whether the message is to be translated from the specified code or into the specified code on physical output. This parameter should be specified only for output to printers that work with ESCAPE sequences not coded according to the EBCDIC core. VTSU ignores these special ESCAPE sequences.

<u>**YES**</u>
VTSU translates the message from or into the specified code. VTSU recognizes and skips the standard ESCAPE sequences.

NO
The control characters SO/SI are evaluated. VTSU does not perform any other code conversion.

CURPOS=
For Extended Line Mode and Mixed Mode, determines whether the cursor position should be returned after input. In Mixed Mode this parameter is accepted only if input and output mode have the value EXTEND. Otherwise it is ignored.

<u>**NO**</u>
No details of the cursor position are supplied. The fields YVTCPOSL and YVTCPOSC are undefined.

YES
The fields YVTCPOSL and YVTCPOSC are returned with the current cursor position (lines and columns) after input.

ENCIN=
Specifies whether or not the next input has to be encrypted (valid only when the connection is established from an emulation supporting encryption).

YES
The next input must be encrypted.

<u>**NO**</u>
The next input must not be encrypted (nevertheless, encryption of the message cannot be prevented the message if encryption has been requested by another means).

ENCOUT=
Specifies if the current output must be encrypted or not (valid only when the connection is established from an emulation supporting encryption).

YES
The output must be encrypted.

<u>**NO**</u>
The output must not be encrypted (nevertheless, encryption of the message cannot be prevented if encryption has been requested by another means).

GETFC=
Specifies whether a function key code is transferred.

<u>**NO**</u>
No function key code is to be transferred.

YES
The logical function key code representing the key that initiates data transfer at the terminal is transferred as the first character of the message.

HCOPY=
Specifies that the message output to a data display terminal is also to be output on a hardcopy device (printer) connected to the terminal.

NO
The message is output to the data display terminal only.

YES
The message output to a data display terminal is also output on a hardcopy device (printer) connected to the terminal.

Notes

- Hardcopy output is performed only if a hardcopy device was assigned to the data display terminal on connection set up or via the MODIFY-TERMINAL-OPTIONS command. For 3270 Data Display Terminals, the hardcopy device must be assigned on connection setup (generated).
- If HCOPY=YES is used with no EXTEND mode, and the message contains the logical control character SPA, EPA, NUM or DAR (if DARPRINTABLE=N), only the last unprotected part of the message is printed, not the entire message.
- If OVERFLOW-CONTROL=NO (MODIFY-TERMINAL-OPTIONS command) is also used, it may be the case that only part of the output is printed on the hardcopy device.

HOM=
(For 816x, 9749, 975x ,9763 and 3270 Data Display Terminals only)

NO
Message output is to be structured and heterogeneous, i.e. each logical line is regarded as a separate output unit.

With 816x, 975x, 9763 and 3270 terminals in operating mode 1 (only for TIAM applications), the effect is that individual logical lines can be modified separately, and so can be specifically transferred back.

YES
Message output is to be unstructured and homogeneous, i.e. the entire message is regarded as a single output unit. The message length is restricted by the size of the output buffer in the system.

With 816x, 975x, 9763 and 3270 terminals in operating mode 1 (only for TIAM applications), the effect is that by modifying a character in an output message, the entire message can be transferred back, so long as it is not explicitly structured by logical display control characters.

IHDR=
Specifies how the message header is to be handled.

YES
The entire message header is passed to the user program (default value for MODE=PHYS).
For 3270 Data Display Terminals, the message header consists of the send key code (AID byte) and the two-byte cursor position.

NO
The message header is not passed to the user program.

INFOLR=
Specifies whether the info line has to be reset.

YES
The info line must be reset.

NO
The info line does not have to be reset.

LOCIN=
Determines how local attributes in the input message are handled.
This operand applies only to data display terminals that support local attributes (e.g. the 9763 Data Display Terminal).

NO
Local attributes are removed from the input message; they are not passed on to the user.

YES
If the input message contains local attributes, they are passed to the user as logical control characters (see the **VTCSET** macro).

LOW=
Determines whether a distinction is to be made between lowercase and uppercase letters.
The default setting for the LOW parameter depends on the MODE operand:

```
MODE=MIXED              the default setting depends on
                        the defined input mode
MODE=PHYS               LOW=YES is the default setting
MODE=LINE/EXTEND/FORM   LOW=NO is the default setting
```

NO
All lowercase letters are transferred as uppercase letters to the user program.

YES
Lowercase letters are also transferred to the user program (default value for MODE=PHYS).

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. PREFIX, MACID and PARAM), see section “S-type macros” on page 29.
The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

NOLOG=
Determines whether logical control characters are to be evaluated.

NO
All logical control characters are evaluated and special physical control characters are accepted (see the **VTCSET** macro, e.g. ESC,DC4).
Other characters < X'40' are replaced by SUB. Printable characters are accepted.

YES
Logical control characters are not evaluated. All characters less than X'40' in EBCDIC code are replaced by SUB (smudge character). Only printable characters are accepted.

NOPOS=
(Only for printers). For Line Mode and Mixed Mode, determines the output location of messages.
In Mixed Mode, this parameter is accepted only if the output mode is MODE=LINE. Otherwise it is ignored.

NO
The output message begins at the start of the next line.

YES
The output message begins at the start of the current line.

OHDR=
Specifies how the user-specific message header is to be handled.

NO
The message header is not prefixed to the output text.

YES
The message contains a user-specific header which the system prefixes to the output text. The length of the message header +1 must be specified in binary form in byte 1 of the message.

Note

When output is to the 8160, 975x and 9763 Data Display terminals and printers locally attached to them, the system (MODE=LINE) or FHS works with parameter specifications (PAG) and does not use a message header (PARAM0, PARAM1). The differences between these two modes are described in the the manuals for data display terminals or printers.

READ=
For Extended Line Mode and Mixed Mode, determines the physical read mode.
In Mixed Mode this parameter is accepted only if input and output mode have the value EXTEND. Otherwise it is ignored.

Note that with DCAM applications in Extended Line Mode, when a YSEND call is followed by a YRECEIVE call, both calls must have the same physical read mode (UNPROT or MODIFIED).

UNPROT
All unprotected fields, including those not, modified, are returned to you. To determine the modified value the data received must be compared with the output data.

MODIFIED
Only the modified fields are returned to you. Each modified field is prefixed in the user buffer with its position on the screen (see logical control characters VPA and HPA).

RETINF=
Determines whether return information is required from printer terminals.

***NONE**
No return information is supplied.

xx
xx = any two printable characters to be supplied with the return information. Single quotes to be returned must be specified twice (e.g. RETINF=''''). The return information is 4 bytes long and has the following structure:

```
Byte 0        Identification (X'41' positive/ X'42' negative)
Bytes 1-2     RETINF byte
Byte 3        Information on printer status (printer-specific)
```

SPECIN=
Requests special input. If special input is to be requested, the SPECIN parameter must be specified beforehand in the requesting output message.

N
Normal input from the terminal.

I
Data is read from the ID card reader. The input data may consist of identity card information or short message K14. This entry is permitted only for the 9749, 975x, 9763, 816x and 3270 Data Display Terminals with a defined ID card reader.
In contrast to TRANSDATA devices, data may be entered on 3270 Data Display Terminals by a defined ID card reader at any time. If entries are requested by the ID card reader, every other entry is converted into K14.

C
The input data is confidential and is to remain invisible on the terminal. This is achieved by blanking or clearing the screen (when the screen format is reset to 24x80) or by overwriting the input line at the printer terminal.

UPDATE=
For Extended Line Mode and Mixed Mode you can determine at the time of format outputs whether the whole screen is created again, or if only the modified lines should be updated. A modified line is one in which either an existing field is updated or a new field is generated. In Mixed Mode, this parameter is accepted only if input and output mode have the value EXTEND. Otherwise it is ignored.

NO
In the case of the first logical new page, the whole screen is created again.

YES
Only the modified lines are updated.
If you create a new field, ensure that the end of this new field has the output attribute 'not visible' and the field attribute 'protected'. If a new field is created, binary nulls are output until the start of the next field. The output attribute 'not visible' suppresses output of binary nulls and blanks are output. The field attribute 'protected' stops the subsequent field being overwritten by the new field. Note that when updating the screen the entire character set (CCSNAME) must be used, as when generating the original screen. Otherwise the original screen is deleted and only the updated lines are output. Also note that field attributes are not implicitly reset when updating.
For example, a field to which the attribute 'premodified' was assigned still has the same attribute even after updating. Attributes must thus be explicitly reset.

Return information and error flags

Standard header:

c	c	b	b	a	a	a	a

A return code relating to the execution of the VTSUCB macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'00'	X'0000'	Successful processing.
X'58'	X'00'	X'0008'	Successful processing, but RETINF byte is incorrect. Parameter ignored.
X'02'	X'00'	X'0008'	Successful processing, but invalid mode for current terminal. Substitute display.
X'00'	X'01'	X'FFFF'	Function not executed. UNIT or FUNCT incorrect.
X'xx' 1)	X'01'	X'0004'	Function not executed. Parameter error in VTSUCB.
X'40'	X'01'	X'0004'	Function not executed. User-specific message header has invalid length.
X'50'	X'01'	X'0004'	Function not executed. User buffer for input has invalid length.
X'60'	X'01'	X'0004'	Function not executed. Requested XHCS function not available. XHCS is not loaded.
X'61'	X'01'	X'0004'	Function not executed. The XHCS function is requested for 7-bit terminals.
X'62'	X'01'	X'0004'	Function not executed. The XHCS function is not supported.
X'80'	X'01'	X'0004'	Function not executed. MODE parameter invalid for command type.
X'86'	X'01'	X'0004'	Function not executed. CCSNAME incompatible with devices.
X'1E'	X'01'	X'0004'	Function not executed. Invalid CCS name.
X'00'	X'03'	X'FFFF'	Function not executed. Incorrect VTSUCB version.
X'xx'	X'20'	X'0004'	Function not executed. Internal error (for diagnostics).
X'00'	X'40'	X'000C'	Output message truncated.
X'00'	X'40'	X'0010'	Input message truncated.
X'00'	X'40'	X'0018'	Extended line mode: input message abbreviated.
X'02'	X'40'	X'0004'	Function not executed. Invalid mode for current terminal. No substitute display.

X'cc'	X'bb'	X'aaaa'	Meaning
X'10'	X'40'	X'0020'	Function not executed. Restricted information.
X'81'	X'40'	X'0004'	Function not executed. No chipcard terminal available for terminal.
X'82'	X'40'	X'0004'	Function not executed. Chipcard terminal available but not accessible.
X'83'	X'40'	X'0004'	Function not executed. Message for chipcard terminal rejected by data display terminal.

Other return codes which, in accordance with conventions, apply to all macros, are given in the table “Standard return codes” on page 43.

[1] The first incorrect parameter in the **VTSUCB** is displayed in Subcode 2.

08:	incorrect length specification in VTSUCB
10:	incorrect MODE operand
11:	incorrect HCOPY operand
12:	incorrect BELL operand
13:	incorrect NOLOG operand
14:	incorrect READ operand
15:	incorrect HOM operand
16:	incorrect RETINF operand
17:	incorrect LOCIN operand
18:	incorrect OHDR operand
19:	incorrect CODETR operand
1A:	incorrect IHDR operand
1B:	incorrect LOW operand
1C:	incorrect SPECIN operand
1D:	incorrect GETFC operand
1E:	incorrect CCSNAME operand
1F:	incorrect CURPOS operand
20:	incorrect UPDATE operand
22:	incorrect AUTOTAB operand
23:	incorrect NOPOS operand
24:	incorrect ENCOUT operand
25:	incorrect ENCIN operand
26:	incorrect INFOLR operand

The leftmost byte of the main code is not currently assigned and is set to the value X'00'. If the standard header contains errors affecting the product identification (e.g. the wrong version), the value is set to X'FF'.

Error messages that are returned to the input/output interface of the access methods when the VTSUCB is not used are supplied in the same form as when the VTSUCB is used. Relevant error information is also returned in the VTSUCB. Error information relating only to the VTSUCB is indicated at the input/output interfaces via a separate return code (X'24') and described in more detail in the VTSUCB return code.

Layout of the DSECT

```
           VTSUCB MF=D,PREFIX=A
1 AVTSUCB    DSECT
1          FHDR  MF=(C,AVTC),EQUATES=NO
2          DS    0A
2 AVTCFHE  DS    0XL8            0   GENERAL PARAMETER AREA HEADER
2 *
2 AVTCIFID DS    0A              0   INTERFACE IDENTIFIER
2 AVTCFCTU DS    AL2             0   FUNCTION UNIT NUMBER
2 *                                  BIT 15    HEADER FLAG BIT,
2 *                                  MUST BE RESET UNTIL FURTHER NOTICE
2 *                                  BIT 14-12 UNUSED, MUST BE RESET
2 *                                  BIT 11-0  REAL FUNCTION UNIT NUMBER
2 AVTCFCT  DS    AL1             2   FUNCTION NUMBER
2 AVTCFCTV DS    AL1             3   FUNCTION INTERFACE VERSION NUMBER
2 *
2 AVTCRET  DS    0A              4   GENERAL RETURN CODE
2 AVTCSRET DS    0AL2            4   SUB RETURN CODE
2 AVTCSR2  DS    AL1             4   SUB RETURN CODE 2
2 AVTCSR1  DS    AL1             5   SUB RETURN CODE 1
2 AVTCMRET DS    0AL2            6   MAIN RETURN CODE
2 AVTCMR2  DS    AL1             6   MAIN RETURN CODE 2
2 AVTCMR1  DS    AL1             7   MAIN RETURN CODE 1
2 AVTCFHL  EQU   8               8   GENERAL OPERAND LIST HEADER LENGTH
2 *
1 *
1 *  SUBCODE 1 VALUES
1 *
1 AVTCECPM     EQU   X'01'         ERROR CLASS PARAMETER ERROR
1 AVTCECIN     EQU   X'20'         ERROR CLASS INTERNAL ERROR
1 AVTCECSP     EQU   X'40'         ERROR CLASS SPECIAL ERROR
1 *
1 *  SUBCODE 2 VALUES
1 *
1 AVTCERLN     EQU   X'08'         ERROR IN LENGTH OF VTSUCB
1 AVTCERMO     EQU   X'10'         ERROR IN MODE PARAMETER
1 AVTCERHC     EQU   X'11'         ERROR IN HARDCOPY PARAMETER
1 AVTCERBE     EQU   X'12'         ERROR IN BELL PARAMETER
1 AVTCERNO     EQU   X'13'         ERROR IN NOLOG PARAMETER
1 AVTCERRD     EQU   X'14'         ERROR IN READ PARAMETER
```

```
1 AVTCERHO     EQU   X'15'         ERROR IN HOM PARAMETER
1 AVTCERRE     EQU   X'16'         ERROR IN RETINF PARAMETER
1 AVTCERLO     EQU   X'17'         ERROR IN LOCIN PARAMETER
1 AVTCEROH     EQU   X'18'         ERROR IN OUTPUT HEADER PARAMETER
1 AVTCERCO     EQU   X'19'         ERROR IN CODETR PARAMETER
1 AVTCERIH     EQU   X'1A'         ERROR IN INPUT HEADER PARAMETER
1 AVTCERLW     EQU   X'1B'         ERROR IN LOWER INPUT PARAMETER
1 AVTCERSP     EQU   X'1C'         ERROR IN SPECIAL INPUT PARAMETER
1 AVTCERGE     EQU   X'1D'         ERROR IN GET FUNCTION CODE PARAMETER
1 AVTCERCC     EQU   X'1E'         INVALID CCSNAME
1 AVTCERCP     EQU   X'1F'         ERROR IN CURPOS PARAMETER
1 AVTCERUP     EQU   X'20'         ERROR IN UPDATE PARAMETER
1 AVTCERWA     EQU   X'21'         ERROR IN WARINFO PARAMETER
1 AVTCERAT     EQU   X'22'         ERROR IN AUTOTAB PARAMETER
1 AVTCERNP     EQU   X'23'         ERROR IN NOPOS PARAMETER
1 AVTCEREO     EQU   X'24'         ERROR IN ENCOUT PARAMETER
1 AVTCEREI     EQU   X'25'         ERROR IN ENCIN PARAMETER
1 AVTCERIR     EQU   X'26'         ERROR IN INFOLR PARAMETER
1 *
1 AVTCERXH     EQU   X'60'         XHCS FCT REQUESTED BUT XHCS NOT LOADED
1 AVTCER7B     EQU   X'61'         XHCS FCT REQUESTED FOR 7-BIT TERMINAL
1 AVTCERBS     EQU   X'62'         XHCS FCT SUPPORT ONLY FROM BS2000 V10
1 *
1 AVTCERIN     EQU   X'7A'         NO PLACE ENOUGH TO INSERT SI/SO CHARS
1 AVTCER1L     EQU   X'7B'         NO PLACE ENOUGH TO INSERT PAR01L CHARS
1 *
1 AVTCERM1     EQU   X'80'         MODE NOT VALID FOR COMMAND TYPE
1 AVTCERM2     EQU   X'81'         MODE=CHIP USED BUT NO CKT ANNOUNCED
1 AVTCERC1     EQU   X'82'         CKT NOT AVAILABLE FROM TERMINAL
1 AVTCERC2     EQU   X'83'         OTHER ERROR CODE FROM DSS BY CKT-MSG
1 AVTCERX2     EQU   X'86'         VTSUCB CCSN INCOMPATIBLE WITH DEVICE
1 AVTCERE1     EQU   X'87'         ENCRYPTION FOR OUTPUT NOT SUPPORTED
1 AVTCERE2     EQU   X'88'         ENCRYPTION FOR INPUT  NOT SUPPORTED
1 AVTCERE3     EQU   X'89'         INFO LINE RESET INVALID WITH MODE
1 *
1 AVTCERO1     EQU   X'40'         HEADER LENGTH OF OUTPUT MSG NOT VALID
1 AVTCERI1     EQU   X'50'         USER BUFFER LEN FOR INPUT NOT VALID
1 *
1 *  MAINCODE VALUES FOR SUBCODE 1 = X'40'
1 *
1 AVTCMRPM     EQU   X'04'         WRONG PARAMETER FOR DEVICE
1 AVTCMROT     EQU   X'0C'         OUTPUT TRUNCATION
1 AVTCMRIT     EQU   X'10'         INPUT TRUNCATION
1 AVTCMRNL     EQU   X'18'         NL IN EXT LINE INPUT MESSAGE
1 *
1 *  MAINCODE VALUES FOR SUBCODE 1 = X'00'
1 *
1 AVTCMRCO     EQU   X'08'         CORRECTED ERROR
```

```
1 *
1 *
1 AVTCLEN      DS    H              LENGTH OF VTSUCB
1 *
1 AVTCINM      DS    C              INPUT MODE FOR MODE=MIXED
1 AVTCOUTM     DS    C              OUTPUT MODE FOR MODE=MIXED
1 *
1         DS    XL4          RETURN INFO (NOT YET USED)
1 *
1 AVTCMODE     DS    C              MODE OF MESSAGE
1 AVTCLINE     EQU   C'L'           LINE MODE
1 AVTCEXT      EQU   C'E'           EXTENDED LINE
1 AVTCINFO     EQU   C'I'           INFO LINE MESSAGE
1 AVTCPHYS     EQU   C'P'           PYHSICAL MODE
1 AVTCTRAN     EQU   C'T'           TRANSPARENT MODE
1 AVTCFORM     EQU   C'F'           FORM MODE
1 AVTCCHIP     EQU   C'C'           CHIPCARD MODE (FOR CKT)
1 AVTCMIXD     EQU   C'M'           MIXED MODE
1 *
1 AVTCHC       DS    C              HARCOPY FUNCTION
1 AVTCHCN      EQU   C'N'           NO HARDCOPY
1 AVTCHCY      EQU   C'Y'           LOCAL/CENTRAL HARDCOPY
1 *
1 AVTCBEL      DS    C              BELL FUNCTION
1 AVTCBELN     EQU   C'N'           NO BELL
1 AVTCBELY     EQU   C'Y'           BELL AFTER OUTPUT
1 *
1 AVTCNLG      DS    C              NO LOG CHARS TO INTERPRET FUNCTION
1 AVTCNLGN     EQU   C'N'           LOGICAL CHARACTERS TO INTERPRET
1 AVTCNLGY     EQU   C'Y'           NO LOGICAL CHARACTERS TO INTERPRET
1 *
1 AVTCRBYT     DS    CL2            RETURN INFO BYTES
1 *
1 AVTCRIN      DS    C              RETURN INFORMATION FUNCTION
1 AVTCRINN     EQU   C'N'           NO RETURN INFORMATION
1 AVTCRINY     EQU   C'Y'           RETURN INFORMATION REQUIRED
1 *
1 AVTCLOC      DS    C              INPUT OF LOCAL CHARACTERS
1 AVTCLOCN     EQU   C'N'           NO LOCAL CHARACTERS REQUIRED
1 AVTCLOCY     EQU   C'Y'           LOCAL CHARACTERS REQUIRED
1 *
1 AVTCOHD      DS    C              OUTPUT HEADER FUNCTION
1 AVTCOHDN     EQU   C'N'           NO OUTPUT HEADER IN USER MSG
1 AVTCOHDY     EQU   C'Y'           OUTPUT HEADER IN USER MESSAGE
1 *
1 AVTCCTR      DS    C              CODE TRANSLATION FUNCTION
1 AVTCCTRN     EQU   C'N'           NO CODE TRANSLATION DONE BY VTSU
1 AVTCCTRY     EQU   C'Y'           CODE TRANSLATION TO/FROM CCS REQ.
```

```
1 *
1 AVTCIHD      DS    C              INPUT HEADER FUNCTION
1 AVTCIHDN     EQU   C'N'           NO INPUT HEADER REQUIRED
1 AVTCIHDY     EQU   C'Y'           INPUT HEADER REQUIRED
1 *
1 AVTCLOW      DS    C              LOWER CHARACTERS FUNCTION
1 AVTCLOWN     EQU   C'N'           TRANSLATE LOWER CHARACTERS
1 AVTCLOWY     EQU   C'Y'           RETAIN LOWER CHARACTERS
1 *
1 AVTCSPIN     DS    C              SPECIAL INPUT FUNCTION
1 AVTCNSPI     EQU   C'N'           NO SPECIAL INPUT
1 AVTCIDIN     EQU   C'I'           INPUT FROM ID-CARD READER
1 AVTCCOIN     EQU   C'C'           CONFIDENTIAL INPUT
1 *
1 AVTCFC       DS    C              FUNCTION CODE
1 AVTCFCN      EQU   C'N'           NO FUNCTION CODE REQUIRED
1 AVTCFCY      EQU   C'Y'           FUNCTION CODE REQUIRED
1 *
1 AVTCHOM      DS    C              HOMOGENEOUS OUTPUT
1 AVTCHOMN     EQU   C'N'           NO HOMOGENEOUS OUTPUT REQUIRED
1 AVTCHOMY     EQU   C'Y'           HOMOGENEOUS OUTPUT REQUIRED
1 *
1 AVTCNOP      DS    C              OUTPUT ON SAME LINE
1 AVTCNOPN     EQU   C'N'           OUTPUT STARTS ON NEXT LINE
1 AVTCNOPY     EQU   C'Y'           OUTPUT STARTS ON CURRENT LINE
1 *
1 AVTCCCNA     DS    CL8            CODED CHARACTER SET NAME
1 *
1 AVTCCUR      DS    C              CURSOR POSITION REQUESTED
1 AVTCCURN     EQU   C'N'           CURSOR POSITION NOT RETURNED
1 AVTCCURY     EQU   C'Y'           CURSOR POSITION GIVEN AFTER INPUT
1 *
1 AVTCPOSL     DS    XL1            CURSOR POSITION (LINE)
1 AVTCPOSC     DS    XL1            CURSOR POSITION (COLUMN)
1 *
1 AVTCREAD     DS    C              READ MODE (EXTENDED LINE MODE)
1 AVTCRDUN     EQU   C'U'           READ UNPROTECTED
1 AVTCRDMO     EQU   C'M'           READ MODIFIED
1 *
1 AVTCUPD      DS    C              SCREEN UPDATE IN EXTENDED LINE MODE
1 AVTCUPDN     EQU   C'N'           NO SCREEN UPDATE -> REFRESH
1 AVTCUPDY     EQU   C'Y'           SCREEN UPDATE
1 *
1 AVTCWAR      DS    C              WAR BYTE REQUESTED
1 AVTCWARN     EQU   C'N'           NO INFO ABOUT WAR BYTE
1 AVTCWARY     EQU   C'Y'           VALUE OF WAR BYTE TO RETURN
1 *
1 AVTCWARI     DS    XL1            RETURNED WAR BYTE VALUE
```

```
1 *
1 AVTCAT       DS    C             AUTOMATIC TABULATION
1 AVTCATS      EQU   C'S'          STANDARD AUTOMATIC TABULATION
1 AVTCATN      EQU   C'N'          AUTOMATIC TABULATION NOT REQUESTED
1 AVTCATY      EQU   C'Y'          AUTOMATIC TABULATION REQUESTED
1 *
1 AVTCEO       DS    C             ENCRYPTION FOR OUTPUT
1 AVTCEON      EQU   C'N'          ENCRYPTION FOR OUTPUT NOT REQUESTED
1 AVTCEOY      EQU   C'Y'          ENCRYPTION FOR OUTPUT REQUESTED
1 *
1 AVTCEI       DS    C             ENCRYPTION FOR INPUT
1 AVTCEIN      EQU   C'N'          ENCRYPTION FOR INPUT NOT REQUESTED
1 AVTCEIY      EQU   C'Y'          ENCRYPTION FOR INPUT REQUESTED
1 *
1 AVTCIR       DS    C             INFO LINE RESET
1 AVTCIRN      EQU   C'N'          INFO LINE RESET NOT REQUESTED
1 AVTCIRY      EQU   C'Y'          INFO LINE REQUESTED
1 *
1              DS    XL1           RESERVED
1 *
1 AVTC#        EQU   *-AVTCFHE         LENGTH OF DSECT
1                *,VTSUCB     350    980309
```

WRCPT – Write checkpoint

General

Application area: Writing checkpoints; see page 162

Macro type: Type S, MF format **1**:
31-bit interface: standard/L/D/E form; see page 29

Macro description

The **WRCPT** macro writes a checkpoint to a specified checkpoint file (cataloged PAM file). The checkpoint comprises ID information, program status, related system status and virtual memory contents. An aborted program can be continued by means of the RESTART-PROGRAM command, making use of a checkpoint (see the “Commands” manual [19]).

Note

- The checkpoint routine opens the file.
- The operand “O” controls whether or not the file is overwritten.
- It is the user's responsibility to branch to the error routine or to a user-defined restart routine.

Macro format and description of operands

```
WRCPT

{ LINK=linkname }
{ FILE=pathname }

[,IDENT=check-id]

,O=NO / YES

,ENDAID=NO / YES

[,MF=L / (E,..) / D]
```

LINK=
Identifies, by means of a link name, the file to which the checkpoint is to be written. The file must be cataloged as a PAM file and entered with this file link name in the TFT (ADD-FILE-LINK command; see the “Commands” manual [19]).

linkname
Link name of the file.

FILE=
Identifies the file to which the checkpoint is to be written. The file must already be cataloged as a PAM file.

pathname
Path name of the file.

IDENT=
Specifies a character string to identify the checkpoint. If this operand is omitted or if the string starts with a blank, the checkpoint is given an ID internally.

check-id
Character string; length = 6 bytes. The first character must not be a blank.

O=
Defines whether or not the specified file is to be (logically) erased before the checkpoint is written.

NO
The specified file is not erased. The checkpoint is written to the end of the file.

YES
The contents of the specified file are erased before the checkpoint is written.

ENDAID=
Specifies whether existing AID connections are to be terminated.

NO
If processing with AID took place before the check point was written, all AID measures remain valid. Writing of the check point is rejected with the return code '68'.

YES
The check point is always written. If processing with AID took place before the check point was written, the connection to AID is terminated. All previously set break points are ineffective after the check point is written.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

Functional description

Upon being invoked, the checkpoint routine validates the operand list. The routine then awaits termination of any outstanding I/O's and determines where in the file the checkpoint is to be written. The checkpoint comprises ID information, program status, related system status and virtual memory contents. All these are required for logically restarting a program at the checkpoint.

When the checkpoint has been successfully completed, a message is logged on SYSOUT for use at restart time. If an error is detected during checkpointing, an error code is returned. This error code is stored in the standard header of the operand list. The first output of the checkpoint routine is a number of PAM blocks with ID information, necessary control blocks, information required for file reopening and the contents of virtual memory. The checkpoint routine outputs a message to SYSOUT if the checkpoint has been completed successfully. This message links a specified ID with a halfpage number for subsequent use in restarting.

The user can reduce the amount of time consumed by the checkpoint routine by controlling the allocation of memory space for the file. The approximate number of PAM blocks used for a single checkpoint may be calculated using the formula

$P = 2n + 12$

where n is the number of pages of virtual memory allocated to the program at checkpoint time. It follows therefore that the checkpoint should be taken at a point in the program where the amount of class 5 and class 6 memory allocated to the program is at a minimum.

Furthermore, the initial allocation of file space should be large enough to accommodate all anticipated checkpoints, thus avoiding secondary allocations. If this is not feasible, the next best alternative is to ensure that the secondary allocation equals or exceeds the requirements of any one checkpoint.

Upon restart of a program at a checkpoint written with **WRCPT**, the program is continued with the next instruction following the **WRCPT** macro. In order to enable the user to check whether the checkpoint routine or the restart routine has been executed, the restart routine sets the secondary return code in byte 5 of the standard header to “R”. Users are thus enabled to decide whether their own restart routine is to be executed.

Notes on the macro call

– If a **WRCPT** macro is specified, the existing copies (S.IN.tsn. ..) of procedure/ENTER files will not be deleted at LOGOFF. These files must exist, otherwise a smooth restart cannot be guaranteed.

- A checkpoint cannot be set:
  - in programs using inter-task communication.
  - in programs that run as shared code, invoke shared code programs or are invoked by shared code programs.
  - in programs which use, directly or indirectly, the memory pool, serialization, ISAM SHARED UPDATE, or UPAM SHARED UPDATE.
  - if the SDF command language is loaded in memory pools in class 5 memory.
  - if RFA connections are open.
  - in programs that process tapes in MAV (multijob processing) mode.
  - in programs that process devices with the ADAM access method.
- The RESTART-PROGRAM command must be issued from within the same computer and device configuration and with the same system as the **WRCPT** macro.
- For file generation groups open at the time of **WRCPT**, the base value between the time of **WRCPT** and the restart time cannot be changed.
- AUDIT functions active at checkpoint time are deactivated.
- The status of user data is **not** automatically restored at restart time. The user must take the necessary steps to ensure that it is restored before restart.
- During checkpoint output, HSMS cannot migrate a user file to a background storage level. The MIGRATE bit in the catalog entry is set to the value INHIBIT (see the “DMS Macros” manual [7]). If the user file is migrated to a background storage level during restart, the RESTART-PROGRAM command with the operand FILE-CHANGE= ALLOWED has to be specified. In this case, the CFID (coded file ID) is not checked.
- If the checkpoint is written in a procedure, the procedure will continue to run as well as the program after the RESTART-PROGRAM command.

Notes on the checkpoint file

- If the checkpoint file resides on a public volume, only the catalog ID of the user's own computer may be specified (in the FCB of the macro); (see also the “HIPLEX MSCF” manual [26]).
- If RFA (remote file access) and SPD (shareable private disk) are being used, the checkpoint file must be cataloged in the computer where the RESTART-PROGRAM command for the file will be issued.
- The checkpoint file may not reside on NK4 pubsets and must have the following attributes:
  BUF-LEN = STD(1)
  BLK-CONTR ≠ DATA
- The checkpoint file may not reside on Net-Storage.

Return information and error flags

Standard header:

c	c	b	b			a	a

A return code relating to the execution of the WRCPT macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aa=Maincode):

X'cc'	X'aa'	Meaning
	X'00'	Checkpoint processed successfully; added note:
X'44'		temporary files were open or link name for temporary job variables exists.
	X'04'	Work area cannot be allocated.
	X'08'	Operand error.
	X'14'	BUF-LEN of the checkpoint file ≠ STD(1)
	X'18'	File opening error; added notes:
X'04'		No work area available.
X'08'		Number of PAM request blocks = 0 (for checkpoint file).
X'10'		Remote or SHARUPD access or eventing not supported with checkpointing.
X'1C'		Error in connection with job variables.
X'20'		Error when validating open FCB.
X'24'		Checkpoint cannot be written (ISAM).
X'28'		Error when waiting for termination of outstanding I/O for a SAM file.
X'2C'		Checkpoint cannot be output (UPAM)
X'30'		Checkpoint cannot be output due to asynchronous I/Os for a BTAM file.
	X'1C'	Checkpoint file is a tape file with LABEL=NSTD or LABEL=NO; no checkpoint.
	X'20'	Error when reading catalog entry for checkpoint file.
	X'24'	Memory pool being used or ISAM file being processed for shared update; no checkpoint.
	X'28'	Serialization being used; no checkpoint.
	X'2C'	Eventing being used; no checkpoint.
	X'30'	Contingency process present; no checkpoint.
	X'34'	DQPAM error; no checkpoint.
	X'38'	End of tape reached during checkpoint output; no checkpoint.
	X'3C'	Program using ADAM; no checkpoint.
	X'40'	Checkpoint output not supported in secure system; no checkpoint.
	X'48'	The checkpoint file has the format BLK-CONTR=DATA; no checkpoint output.
	X'4C'	FASTPAM is still active; no checkpoint output.
	X'50'	Checkpoint file is not a PAM file.
	X'58'	Error when accessing checkpoint file. For details about the error see Subcodes; for an explanation of the error codes see the “Introductory Guide to DMS” [8]. Example: File does not exist at OPEN time (FSTAT error).

X'cc'	X'aa'	Meaning
	X'5C'	Incorrect operand in FILE call for checkpoint file (e.g. SHAREUPD=YES).
	X'60'	The checkpoint file is already open; no checkpoint output.
	X'64'	Macro error during checkpoint output; no checkpoint output.
	X'68'	No checkpoint processing if breakpoints have been defined by means of AID.
	X'6C'	MAREN error; no checkpoint output.
	X'7C'	Data space is used.
	X'76'	POSIX is active.
	X'70'	Internal error during PCB backup; no checkpoint output.
	X'74'	Internal problems with SYSFILE environment; no checkpoint output.

After successful checkpoint processing (X'00') Subcode1 is erased. The restart routine sets the Subcode1 to C'R'.

WRLST – Write record to SYSLST

General

Application area:	Input/output of files and data records; see page 156
Macro type:	Type S, MF format **1**: 24-bit interface: standard/E/L form 31-bit interface: standard/E/L/C/D form; see page 29

The file SYSLST is a temporary (system) file created by the operating system for each task. The contents of SYSLST are output to printer on task termination, and the file is erased. The ASSIGN-SYSLST command gives users the option of assigning their own (cataloged) file, an S variable or a PLAM library element to the SYSLST file.
In contrast to the SYSLST file, the (system) files SYSLST01 to SYSLST99 are only effective when cataloged files have been assigned to them (see also the **SYSFL** macro).

Macro description

The user can employ the **WRLST** macro to write a record to the file SYSLST or to one of the files SYSLST01, ..., SYSLST99. Each macro call transfers a record to the specified file. The entry is made in the SYSLST file, provided that the NUMBER operand is not specified.

Macro format and description of operands

```
WRLST
-------------------------------------------------------------
⎧                                          ⎧ 24 ⎫  ⎫
⎨ record,error[,NUMBER=n][,PARMOD=⎨    ⎬] ⎬
⎪                                          ⎩ 31 ⎭  ⎪
⎩ (1)                                              ⎭
[,MF=(D,pre) / D / I / (E,..) / L / C / (C,pre)]]
```

record

Symbolic address of the record to be written to SYSLST. The record starts with the record length field, followed by print control characters and the data to be transferred.
Format (example):

```
record      DC     Y (recend-record)
            DS     CL2                Reserved bytes
            DC     X'nm'              Print control character
data        DC     C'data-record'     Record to be written
recend      EQU    *
```

Note

The print control character specifies the type of paper feed desired on the printer. The valid print control characters are listed below:

X'4n'	Advance n lines before printing and 1 line after printing. The range of n is $(0 - F)_{16}$ representing up to 16 lines skipped. The setting n = 0 causes a 1-line feed after printing.
X'0n'	Advance n lines after printing. The range of n is $(0 - F)_{16}$ representing a feed of up to 15 lines (n = 0 corresponds to no feed).
X'Cn'	Skip immediately to channel n of the printer carriage control tape. The range of n is $(1 - B)_{16}$.
X'8n'	Print and skip to channel n of the printer carriage control tape. The range of n is $(1 - B)_{16}$.

error

Symbolic address to be branched to if one of the following errors occurs during execution of the **WRLST** macro (see Return information):

- unrecoverable error
- operand error
- truncation error
- memory space saturation

In the event of an error, register R14 contains the address of the next instruction after the **WRLST** macro call. The error code is transferred in register R15.
31-bit interface: If error=0 (address X'00..0') is specified, the program is continued with the instruction following the **WRLST** macro.

NUMBER=

Specifies one of the files SYSLST01 to SYSLST99.

n

(2-digit) number in the range (01,02, ..., 99).

PARMOD=

Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24

The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31

The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

(1)
Register R1 contains the operand list address. The list must be aligned on a word boundary.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A prefix (pre = 1..3 letters) can be specified in the D form of the macro, as shown in the macro format. There is no default value.

The standard header is initialized with MF=I.

If MF=E is specified, the standard header of the operand list is not automatically initialized. The operand list address must be transferred in register R1 and must be aligned on a word boundary. The layout is given below.

Layout of the data area:

Addressing mode	Byte	Contents
24-bit mode	0 1-3 4-7	SYSLSTn (n = number of the SYSLST file). Address of the record to be written (operand “record”). Address to be branched to in the event of errors (operand “error”).
31-bit mode	0-7 8-11 12-15 16	Standard header; format: byte 0-1: X'0024' 2: X'01' 3: X'01' 4-7: return code Address to be branched to in the event of errors (operand “error”). Address of the record to be written (operand “record”). SYSLSTn (number of SYSLST file; operand NUMBER)

Notes on the macro call

- On normal termination, processing is continued with the instruction following the macro expansion. Truncation occurs when the length of the record exceeds 137 or 165 bytes (4-byte length field + 1 byte print control + 132 or 160 characters) in accordance with the printer type specified in the **SYSFL** macro. Register R14 contains the address of the instruction following the macro. Register R15 is unchanged when no error occurs.
- If SYSLST was not cataloged, the following occurs: SYSLST is output to a printer after job termination. After output, the SYSLST file is erased.
- If SYSLST or SYSLSTn is assigned to a cataloged file, the file can also be processed on a long-term basis. In this case, the user is responsible for printing or erasing the file.

Return information and error flags

R15:

0	0	0	0	0	0	a	a

A return code relating to the execution of the WRLST macro is transferred in register R15.

X'aa'	Meaning
X'04'	Unrecoverable error.
X'08'	Operand error.
X'0C'	The record to be written is truncated. Record contents (without record length field) overflow the I/O buffer file 2040 = default value, 2044 when output is to a cataloged file.
X'10'	No more memory space for the file assigned to SYSLSTn. The last PAM page has been allocated and can be written to.

31-bit interface:

No return code is transferred in the standard header. A user dump is produced if the UNIT field of the standard header is supplied with incorrect values.
The two leftmost bytes of register R15 are not used.

If SYSLST is assigned to a cataloged file that does not allow secondary memory space requests (S-ALLOC=0 in the file catalog) and no more memory space is available in this file, the return code X'10' is issued.

WROUT – Write record to SYSOUT

General

Application areas:	Input/output of files and records; see page 156 Data terminal communication; see page 160 Communication; see page 163
Macro type:	Type S, MF format **1**: 24-bit interface: standard/E/L form 31-bit interface: standard/E/L/C/D form; see page 29

- This macro description applies to TIAM V13.2A.
- The following applies when using the 31-bit interface:
  - With MF=C/D, no symbolic names and equates are generated for the standard header. In the event of the operand list being supplied dynamically, the initialization values for the standard header should be taken from an operand list generated with MF=L.
  - No return code is transferred in the standard header.
- The **CUPAB** macro generates a DSECT for the operand list of the **WROUT** macro for 24-bit addressing mode.

Macro description

WROUT sends a message to the SYSOUT file. SYSOUT can be assigned to a PLAM library element, an S variable, a cataloged SAM or ISAM file or typically - to the user's terminal.
If the LOGGING=PARAMETERS (LISTING=YES) operand is specified in the SET-LOGON-PARAMETERS or MODIFY-JOB-OPTIONS command, output to SYSOUT is also routed to SYSLST.

Macro format 1 and description of operands

```
WROUT
-------------------------------------------------------------------------------
record,error[,edit]

 [ { ,MODE=COMP ,OTRSUP={ NO  },OLINEND={ NO  }                      } ]
 [ {                    { YES }         { YES }                      } ]
 [ {                                                                 } ]
 [ {            ,OHCOPY={ NO  },OHDR={ NO  }                         } ]
 [ {                    { YES }      { YES }                         } ]
 [ {                                                                 } ]
 [ { ,MODE=LINE ,OHCOPY={ NO  },OHOM={ NO  }                         } ]
 [ {                    { YES }      { YES }                         } ]
 [ {                                                                 } ]
 [ {            ,OINFO={ NO  },ONOPOSN={ NO  }                       } ]
 [ {                   { YES }         { YES }                       } ]
 [ {                                                                 } ]
 [ {            ,OBELL={ NO  },ONOLOGC={ NO  }                       } ]
 [ {                   { YES }         { YES }                       } ]
 [ {                                                                 } ]
 [ {            ,EXTEND={ NO  }                                      } ]
 [ {                    { YES }                                      } ]
 [ {                                                                 } ]
 [ { ,MODE=PHYS ,OHDR={ NO  },OETB={ NO  }                           } ]
 [ {                  { YES }      { YES }                           } ]
 [ {                                                                 } ]
 [ {            ,OTRANS={ NO  }                                      } ]
 [ {                    { YES }                                      } ]
 [ { ,MODE=FORM                                                      } ]

,RC=OLD / NEW
[,VTSUCBA=addr]
[,ASSIGN=NO /YES]
,PARMOD=24 / 31
[,MF=L / C / (C,pre) / (D,pre) / D / (E,...)]
```

record
Symbolic address of the record to be output. The record starts with the record length field, followed by the print control character and the message to be output.

Record format and example:

Byte 0-1: Length of the message + 4-byte record length field, the length must be > 5.
Byte 2-3: reserved
Byte 4: Print control character; analyzed for printer output only
Byte 5-n: Message

```
record      DC     Y (recend-record)
            DS     CL2               reserved bytes
            DC     X'nm'             print control character
            DC     C'message'        message to be output
recend      EQU    *
```

Note

If SYSOUT is a cataloged file, any records of more than 2044 or 2032 bytes are truncated (register R15=X'0C'), only the first part of the record being written to the file. The different maximum output length depends on the use of the PAM key (FORMAT=NONKEY/KEY).

error

Symbolic address of an error routine in the user program to be branched to in the event of an error.

In the event of an error, register R14 contains the address of the next instruction after the **WROUT** macro call. The error code is transferred in register R15.

31-bit interface: If error = 0 (address X'00..0') is specified, the program is continued with the instruction following the **WROUT** call.

edit

Specifies the edit option for a message to be written to the terminal. The edit option is ignored if SYSOUT is not a terminal. This operand is not necessary if standard functions (all edit bits = 0) are used, if the MODE= operand is specified or if the VTSU control block is used. By direct specification (X'xx') only the first edit byte for output can be set to the meaning described under the **CUPAB** macro.

Notes

This operand continues to be supported for reasons of compatibility only. The edit bytes should be defined via MODE specifications (see the MODE operand) or via the VTSU control block (see the VTSUCBA operand).

MF=

For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A prefix (pre = 1..3 letters) can be specified in the C form and D form of the macro, as shown in the macro format. Default value: pre = CUW

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit addressing).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

The MODE specifications and edit options continue to be supported for compatibility reasons only. They are now summarized in the VTSU control block (VTSUCB, see the **VTSUCB** macro).

MODE=
This operand is not evaluated unless SYSDTA is assigned to the data display terminal.

COMP
Specifies compatible mode. The symbolic operands OTRSUP through OHDR (see below) enable the user program to use all the edit options. Any specifications made directly in the “edit” operand are ignored. Control characters may appear in the output message but these are not checked by the system. This mode is compatible with previous versions of the operating system.
This mode is treated as MODE=LINE for the 8160, 8162, 9749, 975x, 9763, 3270 and X.29 devices. The edit option OLINEND is ignored.
Edit options OTRSUP and OHDR are rejected (RC: X'08').

LINE
Specifies that the current terminal is to be treated as a logical line or page terminal. The message may be structured through the use of logical control characters (see the **VTCSET** macro). If SYSOUT is not a terminal, only logical control characters NL and NP are evaluated, e.g. in the case of output to printer in batch mode.
Any other control characters are invalid and are converted by the system to a user-defined substitute (see the command MODIFY-TERMINAL-OPTIONS SUBSTITUTE-CHARACTER=).

FORM
Specifies format mode. The user program works with the “Terminal Mapping Support” software component (FHS), which edits the message in a form suitable for output to a particular terminal.

PHYS
Specifies that the message is to be output physically at the terminal, i.e. without editing by the system. This permits special device functions to be executed for which the LINE or FORM mode is insufficient. If none of the valid edit options is specified, the system prefixes the message by a standard device message header.

EXTEND=
Determines whether the fields for output texts are to be protected or unprotected.

<u>**NO**</u>
The operator is guided through the system. Only the input request made by the system or the user program is protected against being overwritten.

Output is unprotected and at reduced brightness.

NIL characters in the output text are converted to the substitute characters; in the case of input they are removed.

Depending on the operating mode, the screen may be cleared starting at the cursor when output commences.

YES
(Only for 9749, 975x, 9763, 816x, and 3270 Data Display Terminals)
This entry supports the use of protected and unprotected fields with the aid of logical control characters EPA, NUM and SPA (see the **VTCSET** macro).

Text output is protected and at reduced brightness unless specified otherwise. The message can be structured by means of logical control characters (see **VTCSET**). When using 3270 terminals, note that logical control characters occupy space on the screen. A series of logical control characters, however, requires only one screen position. Areas which the terminal operator can use for input start with EPA or NUM and end with SPA.

During input and output, NIL is treated as a permissible character; it is sent to the data display terminal by the program, and vice versa. When using 3270 terminals, note that the NIL character is not transferred to the processor. Fields that are returned in truncated form are filled with NIL characters up to their original length. This ensures that the fields are always returned to the user in their original (output) length.

The beginning of an output message is displayed at the beginning of the next line following the cursor. If the message does not begin with VPA, the screen is cleared, starting at the cursor and before the first text character.
If the end of the screen is reached during output, output is continued at the top of the screen. This continuation is always unprotected. Overflow control is ineffective here.
Keys RU, EFZ, AFZ and LSP are locked.
Except for OBELL, ILCASE and IGETFC, all other edit options are ignored. See programming notes.

OBELL=
Determines whether an audible signal is provided for output.

NO
No audible signal is provided for output.

YES
An audible signal is heard on output at the end of the message (applies only to 9749, 975x, 9763, 816x and 3270 Data Display Terminals with special hardware feature).

OETB=
Determines the final control character of the output message.

NO
Specifies that the message output at the terminal is to be concluded with the control character ETX.

YES
Specifies that the message output at the terminal is to be concluded with the control character ETB.

OHCOPY=
Determines whether the message output to a data display terminal is also output on a hardcopy device (printer) connected to the terminal.

NO
The message is output via the data display terminal only.

YES
The message output to a data display terminal is also output on a hardcopy device (printer) connected to the terminal. The hardcopy device must be generated or assigned using **TCHNG**.

For 3270 Terminals:

The entire screen contents are output via the hardcopy device. This means that the output may also include previous I/O's. In the event of a series of outputs directly following one another, the hardcopy function is activated for the last output only. Hardcopy output is performed only if a hardcopy device was generated for the terminal when the connection was set up.

There is no hardcopy output if OINFO=YES or EXTEND=YES was specified.

If OHCOPY=YES is used and the message contains one of the logical control characters SPA, EPA, CHS or NUM, only the last unprotected part of the message is output and not the whole message.

If, at the same time, OVERFLOW-CONTROL=NO (MODIFY-TERMINAL-OPTIONS command) was specified, it may be the case that only part of the output is printed on the hardcopy device.

OHDR=
Specifies how the system is to handle the message header.

NO
The message header (in US ASCII code) is prefixed to the output text by the system.

YES
Indicates that the message contains a user-specific header (message header is to be specified in US ASCII code) which the system prefixes to the output text. The length of the message header + 1 must be specified in binary form in byte 5 of the message.

For 3270 Terminals:
The message header is specified in EBCDIC code and consists of the CMD byte and the WCC byte. This message header must be preceded by a byte containing X'01' (length of the TRANSDATA message header + 1).

Note
When output is to the 8160, 975x and 9763 Data Display Terminals and printers locally attached to them, neither the system (MODE=LINE or COMP) nor the terminal mapping support (MODE=FORM) employs any message header (PARAM0, PARAM1). Instead, they work with parameter specifications (PAG). The differences between these two modes are described in the manuals for the data display terminals or printers.

OHOM=
Specifies whether message output is to be structured or homogeneous.

NO
Specifies that the message is to be output, not homogeneously, but in structured form, i.e. one logical line is regarded as the output unit. The message length is unrestricted, provided the logical lines do not exceed 255 characters.

Effect when using mode 1:
Individual logical lines can be separately modified and thus selectively retransferred.

YES
This operand can only be used in conjunction with 816x, 9749, 975x and 9763 Data Display Terminals.
It specifies that the message is to be output homogeneously, in unstructured form, i.e. the entire message is regarded as one output unit. The message length is restricted by the size of the output buffer in the system.

Effect when using mode 1:
By modifying one character in an output message, the entire message can be retransferred, provided the message contains no logical display control characters.

Note
If SYSOUT is assigned to a file, the **WROUT** macro is not executed and SYSLST logging is suppressed.

OINFO=

Determines whether the message is to be output in a special information line.

<u>NO</u>

The message is not output in the special information line.

YES

The message can be mapped to a special information line without destroying important data at the terminal.

The entry is intended particularly for user programs sending messages to terminals “asynchronously” without knowing the current terminal display. Mapping is performed:

– protected, in a hardware display line (e.g. 9749, 9750, 9752 Data Display Terminals), or

– protected, in the last line on the screen (e.g. 816x, 9751, 9753, 3270 Data Display Terminals) if specified in the user program (see the **TCHNG** macro, INFOLIN operand), after previous output with MODE=FORM or MODE=PHYS.

– in all other cases, as a normal line mode message.

If the message length exceeds one screen line, the message is split up and output line by line. The system observes the waiting time specified in the MODIFY-TERMINAL-OPTIONS OVERFLOW-CONTROL=TIME() command.

If OINFO=YES, the OHCOPY=YES entry is ignored, i.e. neither the information line nor the screen contents are printed.

The hardware display line is only reset after the next input followed by an output.

OLINEND=

Specifies how carriage return/line feed characters are to be handled.

<u>NO</u>

Each message output to a terminal always starts on a new line. The necessary control characters are either prefixed to the message by the system or inserted into the message when the physical end-of-line is reached, if the type of terminal requires this to be done.

YES

The message is output to the terminal without the carriage return/line feed control characters supplied by the system. Control must be assumed by the user program.

ONOLOGC=

Specifies whether logical control characters are to be evaluated.

<u>NO</u>

All logical characters are evaluated and special physical control characters are permitted (see the **VTCSET** macro, e.g. ESC, DC4). Any other characters < X'40' are replaced by SUB. Characters ≥ X'40' are accepted.

YES
Logical control characters are not evaluated. All characters < X'40' in EBCDIC code are replaced by SUB. Only characters ≥ X'40' are accepted.

ONOPOSN=
Determines where the message is to start.

<u>**NO**</u>
The message starts at the beginning of the next line.

YES
The message starts at the beginning of the current line (applies only to printer terminals).

OTRANS=
Specifies whether the output data is to be transmitted in standardized or transparent form.

<u>**NO**</u>
Output data is to be transmitted in standardized form, i.e. code conversion takes place.

YES
Output data is to be transmitted in transparent form, i.e. consisting of arbitrary binary characters (5, 7 or 8 bits per character depending on the device code) which are not converted during transmission. If the transmission path was not generated “potentially transparent”, output is rejected with return code X'04'.

OTRSUP=
Specifies whether translation from EBCDIC to device code is suppressed.

<u>**NO**</u>
Translation of the message from EBCDIC to device code is not suppressed, i.e. the program provides the message in EBCDIC and the system translates it into device code.

YES
Translation of the message is suppressed. In this case the program must provide the message in device code.

RC=
Determines where the return code is to be stored.
This operand may be specified only if the 31-bit interface is used.

<u>**OLD**</u>
The return code is stored in the rightmost byte of register R15.

NEW
The return code is stored in both register R15 and the standard header. All 4 bytes of register R15 are allocated for evaluation.
A 4-byte return code is issued only if SYSDTA reads from the data display terminal. In all other cases only a 1-byte return code is issued, irrespective of the return code value.

VTSUCBA=addr
Defines the address of a VTSUCB generated with MF=L.
When using the VTSUCBA operand, the MODE operand and the following edit options are ignored (their value is set to X'FF' in the parameter list). This means that all desired edit options must be specified in the VTSUCB.
This operand may be specified only if the 31-bit interface is used and is not evaluated unless SYSDTA is assigned to a data display terminal. By default, the VTSUCB is not used.

ASSIGN=
Defines whether changes in the SYSOUT assignment are to be displayed.
The user program is notified of the initial default assignment and of each subsequent change to the SYSOUT assignment via the error routine in the user program. When a change to the SYSOUT assignment is detected, the record is not written.
This operand value is permissible only for the 31-bit interface.

NO
Changes to the SYSOUT assignment are not to be displayed.

YES
Changes to the SYSOUT assignment are to be displayed.

Note

The SYSOUT assignment is entered in an output field of the parameter list.
This permits the user program to react to changed general conditions, perform any conversion of the output record which may be required, and then to repeat writing with the corrected record.

Programming notes

for use of the MODE=LINE operand with EXTEND=YES
(for 3270 Data Display Terminals see appendix)

When EXTEND=YES is specified in LINE mode, the user can work with formats without requiring a terminal mapping support component.

- If the user wishes to work with formats, the first output must begin with NP in order to clear the screen and start with text in position 1.1.
- NL positions to the beginning of the next line and clears the remainder of the screen; VPAn positions to the beginning of line n, in which case the remainder of the screen is retained.
- Positioning within a line is only possible with text, spaces or NIL characters.
- After VPAn, NL and CHS, text is protected and displayed with reduced brightness.
- Unprotected fields are output using 'EPA text SPA'.
  Numeric fields are output using 'NUM text SPA'.
- VPAn at the end of the message can be used to place the cursor at the start of the first unprotected field of line n, in which case the screen contents are retained. If no unprotected field starts in line n, the cursor is positioned to the first unprotected field following line n.
  If no VPAn is specified at the end of the message, the cursor is placed in the first unprotected field on the screen.
- Continuation of output/screen update
  NP generates a new screen.
  VPAn at the start of the output changes line n of the screen. VPAn can be used to skip one or more lines. In the current line, a dark area is produced starting at the cursor and extending to the end of the line or a field preceding end-of-line. This is followed by positioning to line n. After completion of the update, the cursor is repositioned using VPAn, otherwise the screen would be cleared starting at the cursor (see above). When NL is used within an updating procedure, the screen is likewise cleared starting at the cursor. If this is to be avoided, VPAn must always be used to jump to a new line.

Macro format 2 and description of operands

WROUT
(1) [,PARMOD=24 / 31

(1)
Register R1 contains the operand list address. The list must be aligned on a word boundary.

PARMOD=
Controls macro expansion; generated as a 24-bit or 31-bit interface.
If PARMOD is omitted, macro expansion corresponds to the specification for the **GPARMOD** macro or to the default value for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

Layout of the data area

Addressing mode	Byte	Contents
24-bit mode	0 1-3 4 5-7	Output edit byte 1 Address of the record to be output (operand “record”) Output edit byte 2 Address to be branched to if an error occurs (operand “error”).
31-bit mode	0-7 8-11 12-15 16 17 18 19 20-23 24-25 26 27	Standard header. For details of the structure, see section “Standard header” on page 43. The initialization values should be taken from an operand list generated with MF=L. No return code is transferred in the standard header if RC=OLD. Address to be branched to if an error occurs (operand “error”). Address of the record to be output (operand “record”) Output edit byte 1 Output edit byte 2 Reserved (X'00') Flag indicating use of VTSUCB and handling of return codes. Address of the VTSUCB Reserved (X'0000') Notification of a change to the SYSOUT assignment is to be provided (input value) Change to the SYSOUT assignment (output value)

When using the 24-bit interface, the values for output edit byte 1/2 should be taken from the table specified with the **CUPAB** macro.
When using the 31-bit interface, they should be taken from a list generated with MF=C/D. This also applies for controlling whether a change to the SYSOUT assignment should be displayed (byte 26) and for the values of the SYSOUT assignment (byte 27).

Values for changing the SYSOUT assignment

Value	Meaning
X'00'	Assignment for SYSOUT not changed
X'01'	SYSOUT is assigned to a file
X'02'	SYSOUT is assigned to a terminal
X'03'	SYSOUT is assigned to an S-variable
X'04'	SYSOUT is assigned to an element of a PLAM library

Note

If SYSOUT is a cataloged file, any records of more than 2044 or 2032 bytes are truncated (RC:X'0C'), only the first part of the record being written to the file. The different maximum output length depends on the use of the PAM key (FORMAT=NONKEY/KEY).
In the case of EAM files, records of more than 2040 bytes are also truncated (return code:X'0C').

Return information and error flags

– During macro processing, register R1 contains the operand list address.

if PARMOD=24:

R15:

0	0	0	0	0	0	a	a

A return code relating to the execution of the WROUT macro is transferred in register R15.

X'aa'	Meaning
X'04'	Unrecoverable error.
X'08'	Operand error.
X'0C'	Output record truncated. Record contents (without record length field) exceed size of I/O buffer: for terminals, this depends on the device type and the network generation; for output to a cataloged file, the relevant value is 2044.
X'10'	During execution of the macro, BREAK was issued on the terminal.
X'20'	Invalid edit option byte: error corrected by the system.
X'38'	Error in connection with POSIX.

if PARMOD=31:

if **RC=OLD:**

Return codes that can occur in addition to those described under PARMOD=24 are

X'aa'	Meaning
X'00'	Normal termination.
X'24'	Error in VTSUCB.

and the return codes which, in accordance with conventions, apply to all macros (see the table “Standard return codes” on page 43).

if **RC=NEW:**

The return codes are entered in both the standard header and register R15.

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the WROUT macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	**X'bb'**	**X'aaaa'**	**Meaning**
X'00'	X'00'	X'0000'	Function processed successfully.
X'00'	X'00'	X'0020'	Function processed successfully; an operand error was corrected by TIAM/VTSU.
X'00'	X'01'	X'0008'	Operand error not corrected.
X'07'	X'01'	X'0008'	Operand error not corrected: the RESERVED fields are not 0
X'00'	X'20'	X'0004'	Internal error.
X'02'	X'20'	X'0004'	Internal error: BCAM message lost.
X'06'	X'20'	X'0004'	Internal error: negative transport acknowledgment.
X'00'	X'20'	X'0028'	Internal error: problems with memory allocation. SYSOUT is assigned to a file whose primary assignment occupies the entire memory and whose secondary assignment is zero.
X'00'	X'40'	X'000C'	Output record truncated.
X'00'	X'40'	X'0010'	BREAK during execution.
X'00'	X'40'	X'0030'	Input/output aborted.
X'01'	X'80'	X'0004'	Internal BCAM bottleneck.
X'09'	X'80'	X'0038'	Error in connection with POSIX: Input/output serialization error.
X'0A'	X'40'	X'0038'	Error in connection with POSIX: If the LOGON task is in system mode, it is not possible to input/output processes generated with fork{}.
		X'24'	VTSU error. In addition to Maincode (rightmost byte), see error information in VTSUCB header.

Example 1

```
  WROUT1   START
           PRINT NOGEN
  WROUT1   AMODE 31
  WROUT1   RMODE 24
           BALR  3,0
           USING *,3
           WROUT MESSAGE,ERROR,PARMOD=31
2                *,@DCEO      999    921011   53531004
           TERM
  ERROR    TERM  DUMP=Y
  *
  MESSAGE  DC    Y(ENDMESS-MESSAGE)      Record length
           DS    CL2                     Reserved
           DC    X'01'                   Print feed control character
           DC    'EXAMPLE WROUT 1'       Text
  ENDMESS  EQU   *
           END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,wrout1), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,wrout1))
%  ASS6011 ASSEMBLY TIME: 293 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 79 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=wrout1, -
/      prog-mode=*any
%  BLS0523 ELEMENT 'WROUT1', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'WROUT1', VERSION ' ' OF '<date> <time>' LOADED
EXAMPLE WROUT 1
```

Example 2

```
  WROUT2   START
           PRINT NOGEN
           BALR  10,0
           USING *,10
           WROUT MESSAGE,ERROR,PARMOD=31,MODE=LINE,OBELL=Y
2                *,@DCEO      999    921011   53531004
  ERROR    NOP   0
           TERM
  *
  MESSAGE  DC    Y(ENDMESS-MESSAGE)
           DS    3X
           DC    C'Output WROUT 2'
  ENDMESS  EQU   *
           END
```

Example 3

Use of VTSUCB

```
  WROUT3   START
           PRINT NOGEN
           BALR  10,0
           USING *,10
           WROUT  MESSAGE,ERROR,PARMOD=31,VTSUCBA=VTSUPAR
  ERROR    NOP   0
           TERM
  *
  VTSUPAR  VTSUCB MODE=LINE,BELL=YES
1                *,VTSUCB     350    980309
  MESSAGE  DC    Y(ENDMESS-MESSAGE)
           DS    3X
           DC    C'Output WROUT 3'
  ENDMESS  EQU   *
           END
```

Example 4

Use of format 2

```
  WROUT4   START
           PRINT NOGEN
           BALR  10,0
           USING *,10
           LA    1,PARAM
           WROUT (1),PARMOD=31
  ERROR    NOP   0
           TERM
  *
  PARAM    WROUT MESSAGE,ERROR,MF=L,PARMOD=31,MODE=LINE,OBELL=Y
2                *,@DCEO      999    921011   53531004
  MESSAGE  DC    Y(ENDMESS-MESSAGE)
           DS    3X
           DC    C'Output WROUT 4'
  ENDMESS  EQU   *
           END
```

WRTRD – Combined input/output

General

Application areas:	Input/output of files and records; see page 156 Data terminal communication; see page 160 Communication; see page 163
Macro type:	Type S, MF format **1**: 24-bit interface: standard/E/L form 31-bit interface: standard/E/L/C/D form; see page 29

- This macro description applies to TIAM V13.2A.
- The following applies when using the 31-bit interface:
  - The C/D form is called with MF=C/D or MF=(C,p)/(D,p).
    p = prefix (up to 3 chars); default value: p = CUB. The prefix modifies field names only (not symbolic names in equates). Any prefix of more than 3 characters is truncated to a length of 3.
  - With MF=C/D, no symbolic names and equates are generated for the standard header. In the event of the operand list being supplied dynamically, the initialization values for the standard header should be taken from an operand list generated with MF=L.
  - No return code is transferred in the standard header.
- The **CUPAB** macro generates a DSECT for the operand list of the **WRTRD** macro for 24-bit addressing mode.

Macro description

WRTRD can be used in timesharing mode only. **WRTRD** writes a record to the terminal and immediately afterwards reads a message from the terminal. Apart from the message written to the terminal no other prompt character appears.

On macro execution, if format 1 is used, the specified operands are stored in an operand table and the start address of this table is loaded into register R1. In the case of format 2 is used, the table specified in the user program is used.

Macro format 1 and description of operands

```
WRTRD
--------------------------------------------------------------------------------------------
record1,[edit1],record2,[edit2],[length],error

 ⎧ ,MODE=COMP ,OTRSUP=⎧ NO ⎫,OLINEND=⎧ NO ⎫,OHDR=⎧ NO ⎫,IHDR=⎧ NO ⎫                      ⎫
 ⎪                    ⎩ YES⎭         ⎩ YES⎭      ⎩ YES⎭      ⎩ YES⎭                      ⎪
 ⎪            ,OHCOPY=⎧ NO ⎫,ILCASE=⎧ NO ⎫,IGETBS=⎧ NO ⎫,ILINEND=⎧ NO ⎫                  ⎪
 ⎪                    ⎩ YES⎭        ⎩ YES⎭        ⎩ YES⎭          ⎩ YES⎭                  ⎪
 ⎪            ,ITRSUP=⎧ NO ⎫                                                             ⎪
 ⎪                    ⎩ YES⎭                                                             ⎪
 ⎪ ,MODE=LINE ,OHCOPY=⎧ NO ⎫,OHOM=⎧ NO ⎫,ONOPOSN=⎧ NO ⎫,IGETIC=⎧ NO ⎫                    ⎪
 ⎪                    ⎩ YES⎭      ⎩ YES⎭         ⎩ YES⎭        ⎩ YES⎭                    ⎪
[⎨            ,OBELL=⎧ NO ⎫,ONOLOGC=⎧ NO ⎫,EXTEND=⎧ NO ⎫,IGETFC=⎧ NO ⎫                    ⎬]
 ⎪                   ⎩ YES⎭         ⎩ YES⎭        ⎩ YES⎭        ⎩ YES⎭                    ⎪
 ⎪            ,ILCASE=⎧ NO ⎫,IGETBS=⎧ NO ⎫,ICFD=⎧ NO ⎫                                   ⎪
 ⎪                    ⎩ YES⎭        ⎩ YES⎭      ⎩ YES⎭                                   ⎪
 ⎪ ,MODE=FORM ,IGETBS=⎧ NO ⎫,ILCASE=⎧ YES⎫                                               ⎪
 ⎪                    ⎩ YES⎭        ⎩ NO ⎭                                               ⎪
 ⎪ ,MODE=PHYS ,OHDR=⎧ NO ⎫,OTRANS=⎧ NO ⎫,OETB=⎧ NO ⎫,ITRSUP=⎧ NO ⎫                       ⎪
 ⎪                  ⎩ YES⎭        ⎩ YES⎭      ⎩ YES⎭        ⎩ YES⎭                       ⎪
 ⎪            ,IHDR=⎧ YES⎫,ILCASE=⎧ NO ⎫,IGETBS=⎧ NO ⎫                                   ⎪
 ⎩                  ⎩ NO ⎭        ⎩ YES⎭        ⎩ YES⎭                                   ⎭

,RC=OLD / NEW
[,VTSUCBA=addr]
[,TIMER=value]
,PARMOD=24 / 31
[,MF=L / C / (C,pre) / (D,pre) / D / (E,...)]
```

record1

Symbolic address of the record to be output. The record starts with the record length field, followed by one (any) character and the message to be output.

Record format and example:

Byte 0-1: Length of the message + 4-byte record length field
Byte 2-3: reserved
Byte 4: Any character; neither transferred nor analyzed
Byte 5-n: Data record

```
RECORD      DC     Y (RECEND–RECORD)
            DS     CL2              reserved bytes
            DC     X'00'
            DC     C'DATA–RECORD'   record to be transferred
RECEND      EQU    *
```

edit1

This operand specifies the edit option for the record to be written. Direct specification (X'xx') permits only the first edit byte for output to be set to the meaning specified in the **CUPAB** macro description. This operand is not required when default functions (all edit bits = 0) are used, when the MODE operand is specified or when the VTSU control block is used.

record2

Symbolic address of a field to which the record from the terminal is to be transferred. The record is read as a variable-length record (the first 4 bytes specifying the record length).

Record format and example:

Byte 0-1: Length of the message + 4-byte record length field
Byte 2-3: reserved
Byte 4-n: Data record

```
RECORD2    DS    0CL74
LENGTH     DS    CL2
RESERV     DS    CL2
DATA       DS    CL70
```

edit2

Specifies the edit option for the record to be read. Direct specification (X'xx') permits only the first edit byte for output to be set to the meaning described in the **CUPAB** macro description. This operand is not required when default functions (all edit bits = 0) are used, when the MODE operand is specified or when the VTSU control block is used.

Note

> Operands edit1 and edit2 continue to be supported for reasons of compatibility only. Edit bytes should be controlled via MODE specifications or via the VTSU control block (VTSUCBA operand).

length
Length of the field specified with “record2” (including 4-byte record length field);
5 ≤ length ≤ 32767.
If this operand is omitted, the length attribute of the specified field is assumed.

error
Symbolic address to be branched to in the event of an error.
In the event of an error, register R14 contains the address of the next instruction after the **WRTRD** macro call. The error code is transferred in register R15.
31-bit interface: If error = 0 (address X'00..0') is specified, the program is continued with the instruction following the **WRTRD** macro.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.
A prefix (pre = 1..3 letters) can be specified in the C form and D form of the macro, as shown in the macro format. Default value: pre = CUB

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

The MODE specifications and edit options continue to be supported for compatibility reasons only. They are now summarized in the VTSU control block (VTSUCB, see the **VTSUCB** macro).

MODE=COMP
Specifies compatible mode. The symbolic operands OTRSUP through OPTAPE (see below) enable the user program to use all the edit options. Any specifications made directly in the “edit” operand are ignored. Control characters may appear in the output message but these are not checked by the system. This mode is compatible with previous versions of the operating system.
This mode is treated as MODE=LINE for the 8160, 8162, 9749, 975x, 9763, 3270 and X.29 devices. Edit options OLINEND and ILINEND are ignored. Edit options OTRSUP, OHDR, ITRSUP and IHDR are rejected (RC: X'08').

LINE
The current terminal is to be treated as a logical line or page terminal. The message may be structured through the use of logical control characters (see the **VTCSET** macro).
Any other control characters are invalid for output and are converted by the system to a user-defined substitute character (see the command MODIFY-TERMINAL-OPTIONS SUBSTITUTE-CHARACTER=). If SYSOUT is not a terminal, only logical control characters NL and NP are evaluated, e.g. for output to printers in batch mode.
The device-specific message header is not supplied during input.

FORM
Specifies format mode. The user program works with the “Terminal Mapping Support” software component (FHS), which edits the message in a form suitable for output to a particular terminal.

PHYS
The message is to be output to or read from the terminal physically, i.e without editing by the system. This permits special device functions to be executed for which the LINE or FORM mode is insufficient. If none of the valid edit options is specified, the system prefixes the output message by a standard device message header; the device message header is not removed from the input message. Lowercase letters are converted to uppercase letters and a backspace function may be executed.

EXTEND=
Determines whether the fields for output texts are to be protected or unprotected.

NO
The operator is guided through the system. Only the input request made by the system or the user program is protected against being overwritten.

Output is unprotected and at reduced brightness.

NIL characters in the output text are converted to the substitute characters; in the case of input they are removed.

Depending on the operating mode, the screen may be cleared starting at the cursor when output commences.

YES
(Only for 9749, 975x, 9763, 816x, and 3270 Data Display Terminals)
This entry supports the use of protected and unprotected fields with the aid of logical control characters EPA, NUM and SPA (see the **VTCSET** macro).
Text output is protected and at reduced brightness unless specified otherwise. The message can be structured by means of logical control characters (see **VTCSET**).
When using 3270 terminals, note that logical control characters occupy space on the screen. A series of logical control characters, however, requires only one screen position. Areas which the terminal operator can use for input start with EPA or NUM and end with SPA.

During input and output, NIL is treated as a permissible character; it is sent to the data display terminal by the program, and vice versa. When using 3270 terminals, note that the NIL character is not transferred to the processor. Fields that are returned in truncated form are filled with NIL characters up to their original length. This ensures that the fields are always returned to the user in their original (output) length.

The beginning of an output message is displayed at the beginning of the next line following the cursor. If the message does not begin with VPA, the screen is cleared, starting at the cursor and before the first text character.
If the end of the screen is reached during output, output is continued at the top of the screen. This continuation is always unprotected. Overflow control is ineffective here.
Keys RU, EFZ, AFZ and LSP are locked.
Except for OBELL, ILCASE and IGETFC, all other edit options are ignored.
If the NL control character is recognized in an input message, processing continues and the return code X'2C' is issued.

ICFD=
Specifies whether confidential data is to be protected.

NO
No precautions are to be taken to protect confidential data.

YES
The input data is confidential and is to remain invisible on the terminal. This is achieved by blanking or clearing the screen or by overwriting the input line at the printer terminal.

IGETBS=
Determines whether underline characters (X'6D') are to be passed to the user program. This operand should only be specified for 8103 Data Display Terminals.

NO
Underline characters are not passed to the user program. Instead, the system performs the correction function

YES
The underline characters (X'6D') are passed to the user program without being evaluated by the system.

IGETFC=
Determines whether a function key code is transferred.

NO
No function key code is to be transferred.

YES
The first byte of the input area is to contain the standardized function key code. This code identifies the terminal key used to initiate data transfer. A table with the standard function key codes is included in the appendix on page 1153.

IGETIC=
Determines whether the input source is to be changed.

NO
The input source is not to be changed.

YES
Data is to be entered from the connected ID card reader. The input data may consist of identity card information or short message K14. This entry is permitted only for the 9749, 975x 9763, 816x and 3270 Data Display Terminals with a defined ID card reader (see also the **TSTAT** macro, TYPE=TCHAR).
In contrast to TRANSDATA devices, data may be entered on 3270 Data Display Terminals by a defined ID card reader at any time. If entries are requested by the ID card reader every other entry is converted into K14.

Note

The IGETIC operand is ignored if the ICFD operand is also specified or if no ID card reader is connected. The input source remains unchanged.

IHDR=
Specifies how the message header is to be handled.

NO
The message header is not transferred to the user program.

YES
The entire message header is transferred to the user program.
With 3270 terminals, the message header consists of the application ID (AID byte) and the two-byte cursor position.

ILCASE=
Specifies whether a distinction is to be made between uppercase and lowercase letters.

NO
All lowercase letters are transferred to the user program as uppercase letters.

YES
Lowercase letters are also transferred to the user program.

ILINEND=
Specifies how carriage return/line feed characters are to be handled.

NO
The carriage return and line feed characters are not passed to the user program.

YES
The carriage return and line feed characters are passed to the user program.

ITRSUP=
Specifies whether the translation of device code to EBCDIC is to be suppressed.

<u>**NO**</u>
Translation from device code to EBCDIC is not suppressed. The user program receives the message in EBCDIC.

Exception
On the 816x, 9749, 975x and 9763 Data Display Terminals the message header is always supplied in device code.

YES
Translation from device code to EBCDIC is suppressed. The user program receives the message in device code.

OBELL=
Determines whether an audible signal is provided on output.

<u>**NO**</u>
No audible signal is provided for output.

YES
An audible signal is heard on output at the end of the message (applies only to 9749, 975x, 9763, 816x and 3270 Data Display Terminals with special hardware feature).

OETB=
Determines the final control character of the output message.

<u>**NO**</u>
Specifies that the message output at the terminal is to be concluded with the control character ETX.

YES
Specifies that the message output at the terminal is to be concluded with the control character ETB.

OHCOPY=
Determines whether the message output to a data display terminal is also output on a hardcopy device (printer) connected to the terminal.

<u>**NO**</u>
The message is output via the data display terminal only.

YES
The message output to a data display terminal is also output on a hardcopy device (printer) connected to the terminal. The hardcopy device must be generated or assigned using the MODIFY-TERMINAL-OPTIONS command.

For 3270 Terminals:

The entire screen contents are output via the hardcopy device. This means that the output may also include previous I/O's. In the event of a series of outputs directly following one another, the hardcopy function is activated for the last output only. Hardcopy output is performed only if a hardcopy device was generated for the terminal when the connection was set up.

There is no hardcopy output if EXTEND=YES or MODE=EXTEN was specified.
If OHCOPY=YES is used and the message contains one of the logical control characters SPA, EPA, CHS or NUM, only the last unprotected part of the message is output and not the whole message. If, at the same time, OVERFLOW-CONTROL=NO (MODIFY-TERMINAL-OPTIONS command) was specified, it may be the case that only part of the output is printed on the hardcopy device.

OHDR=
Specifies how the system is to handle the message header.

NO
The message header (in US ASCII code) is prefixed to the output text by the system.

YES
Indicates that the message contains a user-specific header (message header is to be specified in US ASCII code) which the system prefixes to the output text. The length of the message header + 1 must be specified in binary form in byte 1 of the message.
For 3270 Terminals:

The message header is specified in EBCDIC code and consists of the CMD byte and the WCC byte. This message header must be preceded by a byte containing X'01' (length of the TRANSDATA message header + 1).

Note

When output is to the 8160, 975x and 9763 Data Display Terminals and printers locally attached to them, neither the system (MODE=LINE or COMP) nor the terminal mapping support (MODE=FORM) employs any message header (PARAM0, PARAM1). Instead, they work with parameter specifications (PAG). The differences between these two modes are described in the manuals for data display terminals or printers.

OHOM=
Specifies whether message output is to be structured or homogeneous.

NO
Specifies that the message is to be output, not homogeneously, but in structured form, i.e. one logical line is regarded as the output unit. The message length is unrestricted, provided the logical lines do not exceed 255 characters.
Effect when using 816x, 975x, 9763 and 3270 Data Display Terminals in mode 1: Individual logical lines can be separately modified and thus selectively retransferred.

YES
This operand can only be used in conjunction with 816x, 9749, 975x, 9763 and 3270 Data Display Terminals.

The message is to be output homogeneously, in unstructured form, i.e. the entire message is regarded as a single output unit. The message length is restricted by the size of the output buffer in the system.

Effect when using 816x, 975x, 9763 and 3270 Data Display Terminals in mode 1:
By modifying one character in an output message, the entire message can be retransferred, provided the message is not explicitly structured by logical display control characters.

OLINEND=
Specifies how carriage return/line feed characters are to be handled.

NO
Each message output to a terminal always starts on a new line. The necessary control characters are either prefixed to the message by the system or inserted into the message when the physical end of line is reached, if the type of terminal requires this to be done.

YES
The message is output to the terminal without the carriage return/line feed control characters supplied by the system. Control must be assumed by the user program.

ONOLOGC=
Specifies whether logical control characters are to be evaluated.

NO
All logical characters are evaluated and special physical control characters are permitted (see the **VTCSET** macro, e.g. ESC, DC4).
Any other characters < X'40' are replaced by SUB. Characters ≥ X'40' are accepted.

YES
Logical control characters are not evaluated. All characters < X'40' in EBCDIC code are replaced by SUB. Only characters ≥ X'40' are accepted.

ONOPOSN=
Determines where the message is to start.

NO
The message starts at the beginning of the next line.

YES
The message starts at the beginning of the current line (applies only to printer terminals).

OTRANS=
Specifies whether the output data is to be transmitted in standardized or transparent form.

<u>**NO**</u>
Output data is to be transmitted in standardized form, i.e. code conversion takes place.

YES
Output data is to be transmitted in transparent form, i.e. consisting of arbitrary binary characters (5, 7 or 8 bits per character depending on the device code) which are not converted during transmission. If the transmission path was not generated “potentially transparent”, output is rejected with return code X'04'.

OTRSUP=
Specifies whether translation from EBCDIC to device code is suppressed.

<u>**NO**</u>
Translation of the message from EBCDIC to device code is not suppressed, i.e. the program provides the message in EBCDIC and the system translates it into device code.

YES
Translation of the message is suppressed. In this case the program must provide the message in device code.

RC=
Determines where the return code is to be stored.
This operand may be specified only if the 31-bit interface is used.

<u>**OLD**</u>
The return code is stored in the rightmost byte of register R15.

NEW
The return code is stored in both register R15 and the standard header. All 4 bytes of register R15 are allocated for evaluation.
A 4-byte return code is issued only if SYSDTA reads from the data display terminal. In all other cases only a 1-byte return code is issued, irrespective of the return code value.

TIMER=
Defines a maximum waiting time for input. If no input is received within the defined waiting time, a return code is issued. This operand may only be specified if the 31-bit interface is used.

value
Waiting time between 10 and 3600 seconds. The default value is UNLIMITED, i.e. no timer is used.

VTSUCBA=
Defines the address of a VTSUCB generated with MF=L.
When using the VTSUCBA operand, the MODE operand and the following edit options are ignored (their value is set to X'FF' in the parameter list). This means that all desired edit options must be specified in the VTSUCB.
This operand may be specified only if the 31-bit interface is used.
By default, the VTSUCB is not used.

addr
Symbolic address (name) of the VTSUCB.

Programming notes

for the use of the MODE=EXTEND or EXTEND=YES operand
(for 3270 Data Display Terminals see appendix)

When MODE=EXTEND or EXTEND=YES is specified in LINE mode, the user can work with formats without requiring a terminal mapping support component.

– If the user wishes to work with formats, the first output must begin with NP in order to clear the screen and start with text in position 1.1.
– NL positions to the beginning of the next line and clears the remainder of the screen; VPAn positions to the beginning of line n, in which case the remainder of the screen is retained.
– Positioning within a line is only possible with text, spaces or NIL characters.
– After VPAn, NL and CHS, text is protected and displayed with reduced brightness.
– Unprotected fields are created using 'EPA text SPA'.
Blanked, unprotected fields are created using 'DAR text SPA'. Numeric fields are created using 'NUM text SPA'.
– VPAn at the end of the message can be used to place the cursor at the start of the first unprotected field of line n, in which case the screen contents are retained. If no unprotected field starts in line n, the cursor is positioned to the first unprotected field following line n.
If no VPAn is specified at the end of the message, the cursor is placed in the first unprotected field on the screen.
– Continuation of output/screen update
NP generates a new screen.
VPAn at the start of the output changes line n of the screen. VPAn can be used to skip one or more lines. In the current line, a dark area is produced starting at the cursor and extending to the end of the line or a field preceding end-of-line. This is followed by positioning to line n. After completion of the update, the cursor is repositioned using VPAn, otherwise the screen would be cleared starting at the cursor (see above). When NL is used within an updating procedure, the screen is likewise cleared starting at the cursor. If this is to be avoided, VPAn must always be used to jump to a new line.

Macro format 2 and description of operands

WRTRD
(1)

(1)
Register R1 contains the operand list address. The list must be aligned on a word boundary.

Layout of the data area

Addressing mode	Byte	Contents
24-bit mode	0 1-3 4 5-7 8 9 10-11 12-15	Output edit byte 1 Address of the record to be output (operand “record1”) Input edit byte 1 Address of the field to which the record is to be transferred (operand “record2”) Output edit byte 1 Input edit byte 2 Maximum length of the record to be read (operand “length”). Address to be branched to if an error occurs (operand “error”).
31-bit mode	0-7 8-11 12-15 16-19 20 21 22 23 24-25 26 27 28-31 32-33 34-35	Standard header. For details of the structure, see section “Standard header” on page 43. The initialization values should be taken from an operand list generated with MF=L. No return code is transferred in the standard header if RC=OLD. Address to be branched to if an error occurs (operand “error”). Address of the record to be output (operand “record1”) Address of the field to which the record is to be transferred (operand “record2”) Output edit byte 1 Output edit byte 2 Input edit byte 1 Input edit byte 2 Maximum length of the record to be read(operand “length”). Reserved (X'00') Flag indicating use of VTSUCB and return code handling Address of the VTSUCB Values for timer Reserved (X'00000000')

When using the 24-bit interface, the values for input/output edit byte 1/2 should be taken from the table specified with the **CUPAB** macro; when using the 31-bit interface, they should be taken from a list generated with MF=C/D.

When “BREAK” occurs during a write/read operation, the program count is reset to the beginning of the macro expansion so that, after the interrupt is processed, the macro is repeated.
If the length of the record written (minus four bytes for the length field and one byte for the reserved byte) exceeds the terminal buffer size, the record is truncated. The user receives control at the error address with error code X'10' in register R15.
If the size of a record read exceeds the designated length (minus four bytes for the length field), the record is truncated. A branch is made to the error address of the user program and error code X'OC' is transferred in register R15.

Return information and error flags

Whenever possible, the system corrects any edit options which are invalid for a particular device or for the MODE selected (return code X'20'). During macro processing, register R1 contains the operand list address.

if PARMOD=24:

R15:	0	0	0	0	0	0	a	a

A return code relating to the execution of the WRTRD macro is transferred in register R15.

X'aa'	Meaning
X'00'	Function executed successfully.
X'04'	Unrecoverable error.
X'08'	Operand error.
X'0C'	Read truncation. Record length exceeds specified length. A header is prefixed to the input message by the MSV terminal. If the message has already reached the length of the system buffer, the message is truncated as a result of this header.
X'10'	Write truncation. The length of the output record exceeds the size of the terminal buffer. Excess characters are not output.
X'14'	WRTRD was called in a batch job.
X'18'	End of input (ETX)
X'20'	Invalid edit option byte, corrected by system.
X'2C'	Input starts with control character NL (only with edit option EXTEND=YES). Input length is truncated in 3270.
X'38'	Error in connection with POSIX.

if PARMOD=31:

if **RC=OLD**:

Return codes that can occur in addition to the return codes described under PARMOD=24

X'aa'	**Meaning**
X'24'	Error in VTSUCB.

and the return codes which, in accordance with conventions, apply to all macros (see the table “Standard return codes” on page 43).

if **RC=NEW**:

The return codes are entered in both the standard header and register R15.

Standard header:

c	c	b	b	a	a	a	a

The following return code relating to the execution of the WRTRD macro is transferred in the standard header (cc=Subcode2, bb=Subcode1, aaaa=Maincode):

X'cc'	**X'bb'**	**X'aaaa'**	**Meaning**
X'00'	X'00'	X'0000'	Function processed successfully.
X'00'	X'00'	X'0020'	Function processed successfully; an operand error was corrected by TIAM/VTSU.
X'00'	X'01'	X'0008'	Operand error not corrected.
X'07'	X'01'	X'0008'	Operand error not corrected: the RESERVED fields are not 0.
X'08'	X'01'	X'0008'	Operand error not corrected: the value of the TIMER operand is not within the permitted range of 10 through 3600 seconds.
X'00'	X'20'	X'0004'	Internal error.
X'02'	X'20'	X'0004'	Internal error: BCAM message lost.
X'05'	X'20'	X'0004'	Internal error: input message too long.
X'06'	X'20'	X'0004'	Internal error: negative transport acknowledgment.
X'00'	X'40'	X'0004'	Input/output aborted.
X'00'	X'40'	X'000C'	Input record length > specified length: input record was truncated.
X'00'	X'40'	X'0010'	Output record truncated.
X'00'	X'40'	X'0014'	BREAK in WRTRD. If “BREAK” occurs during a read/write operation and RC=NEW, the user is supplied with the return code '00400014'. If RC=OLD, the TU program count is reset to the beginning of the macro expansion so that, after the interrupt is processed, the macro can be repeated.

X'cc'	X'bb'	X'aaaa'	Meaning
X'00'	X'40'	X'0018'	End of input.
X'00'	X'40'	X'002C'	NL detected (this return code can only occur for edit options with EXTEND=YES).
X'00'	X'40'	X'0034'	Timeout (no input received within the defined waiting time).
X'00'	X'00'	X'0014'	SYSFILE error: WRTRD in batch mode.
X'01'	X'80'	X'0004'	Internal BCAM bottleneck.
X'09'	X'80'	X'0038'	Error in connection with POSIX: Input/output serialization error.
X'0A'	X'40'	X'0038'	Error in connection with POSIX: If the LOGON task is in system mode, no inputs/outputs of processes generated with fork{} are possible.
		X'24'	VTSU error. In addition to the main code (rightmost byte), see error information in VTSUCB header.

Example 1

The example effects the output of a message to which the user must respond. The query is repeated as long as 'N' is entered as the response. If the response is ≠ 'N' the program is terminated. The **WRTRD** macro is called in macro call format 1.

```
  WRTRD1   START
           PRINT NOGEN
  WRTRD1   AMODE 31
  WRTRD1   RMODE 24
           BALR  3,0
           USING *,3
  QUEST    WRTRD QUERY,,INPUT,,5,END,PARMOD=31
2                *,@DCEO      999    921011   53531004
2                *,@DCEI      999    921011   53531002
           CLI   REPLY,'N'
           BE    QUEST
  END      WROUT TEXT,TERM,PARMOD=31
2                *,@DCEO      999    921011   53531004
  TERM     TERM
  **** Definitions ****
  QUERY    DC    Y(ENDQU-QUERY)
           DS    CL2
           DC    X'01'
           DC    'TERMINATE PROGRAM (Y/N) ?'
  ENDQU    EQU   *
  INPUT    DS    0CL5
           DS    CL4
  REPLY    DS    CL1
  TEXT     DC    Y(ENDTEXT-TEXT)
           DS    CL3
           DC    C'*** The WRTRD1 program was terminated. ***'
  ENDTEXT  EQU   *
           END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,wrtrd1), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,wrtrd1))
%  ASS6011 ASSEMBLY TIME: 310 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 82 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=wrtrd1, -
/      prog-mode=*any
%  BLS0523 ELEMENT 'WRTRD1', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'WRTRD1', VERSION ' ' OF '<date> <time>' LOADED
TERMINATE PROGRAM (Y/N) ?
n
TERMINATE PROGRAM (Y/N) ?
y
*** The WRTRD1 program was terminated. ***
```

Example 2

This example effects the same procedure as Example 1, but the **WRTRD** macro is called in macro call format 2.

```
  WRTRD2   START
           PRINT NOGEN
           BALR  10,0
           USING *,10
  LOOP     LA    1,PARAM
           WRTRD (1),PARMOD=31
           CLI   REPLY,'N'
           BE    LOOP
  END      WROUT TEXT,TERM,PARMOD=31
2                *,@DCEO      999    921011   53531004
  TERM     TERM
  **** Definitions ****
           DS    0F
  PARAM    WRTRD QUERY,,INPUT,,5,END,MF=L,PARMOD=31
2                *,@DCEO      999    921011   53531004
2                *,@DCEI      999    921011   53531002
  *
  QUERY    DC    Y(ENDQU-QUERY)
           DS    3X
           DC    'TERMINATE PROGRAM (Y/N) ?'
  ENDQU    EQU   *
  INPUT    DS    0CL5
  LENGTH   DS    CL2
  UNUSED   DS    CL2
  REPLY    DS    CL1
  TEXT     DC    Y(ENDTEXT-TEXT)
           DS    CL3
           DC    C'*** The WRTRD2 program was terminated. ***'
  ENDTEXT  EQU   *
           END
```

Example 3

```
  WRTRD3   START
           PRINT NOGEN
           BALR  10,0
           USING *,10
           WRTRD MESSAGE,,INPUT,,40,ERROR,PARMOD=31,MODE=LINE,          C
                 OBELL=Y,ILCASE=Y,ICFD=Y
2                *,@DCEO      999    921011   53531004
2                *,@DCEI      999    921011   53531002
  ERROR    NOP   0
           TERM
  **** Definitions ****
  MESSAGE  DC    Y(ENDMESS-MESSAGE)
           DS    3X
           DC    C'Output WRTRD example 3'
  ENDMESS  EQU   *
  INPUT    DS    0CL14
  LENGTH   DS    CL2
  UNUSED   DS    CL2
  DATA     DS    CL10
           END
```

Example 4

Use of VTSUCB

```
  WRTRD4   START
           PRINT NOGEN
           BALR  10,0
           USING *,10
           WRTRD MESSAGE,,INPUT,,40,ERROR,PARMOD=31,VTSUCBA=VTSUPAR
  ERROR    NOP   0
           TERM
  *
  VTSUPAR  VTSUCB MODE=LINE,BELL=YES,LOW=YES,SPECIN=C
1                *,VTSUCB     350    980309
  MESSAGE  DC    Y(ENDMESS-MESSAGE)
           DS    3X
           DC    C'Output WRTRD example 4'
  ENDMESS  EQU   *
  INPUT    DS    0CL40
  LENGTH   DS    CL2
  UNUSED   DS    CL2
  DATA     DS    CL36
           END
```

Example 5

WRTRD with extended line mode

```
  WRTRD5   START
           PRINT GEN
           BALR  3,0
           USING *,3
  QUEST    WRTRD OUTPUT,,INPUT,,90,END,MODE=LINE,EXTEND=YES,PARMOD=31
1 QUEST    ##SPASS S0001S,S0001D                                       A312
2          CNOP  0,4
2 QUEST    BAS   1,S0001S          ADDRESS AND SKIP PARAMS
1 S0001D     DS OF                                                     A340
1          FHDR  UNIT=36,FUNCT=19,VERS=2
2          DS    0A
2          DS    0XL8               GENERAL OPERAND LIST HEADER
2          DC    AL2(36)            FUNCTION UNIT NUMBER
2          DC    AL1(19)            FUNCTION NUMBER
2          DC    AL1(2)             FUNCTION INTERFACE VERSION NUMBER
2          DC    X'FFFFFFFF'         Returncode is virgin
1          DC    A(END)                  ERROR RETURN ADDRESS
1          DC    AL4(OUTPUT)             MESSAGE AREA ADDRESS
1          DC    AL4(INPUT)              READ IN AREA DDRESS
1          DS    AL1(0)                  PLACE FOR O.EDIT BYTE 1
1          DS    AL1(0)                  PLACE FOR O.EDIT BYTE 2
1          DS    AL1(0)                  PLACE FOR I.EDIT BYTE 1
1          DS    AL1(0)                  PLACE FOR I.EDIT BYTE 2
1          DC    AL2(90)                 NUMBER OF CHARS. TO BE READ
1          DC    AL1(0)                  RESERVED 2
1          DC    AL1(0)                  FLAG BYTE 1
1          DC    AL4(0)                  VTSUCB ADDRESS
1          DC    AL2(0)                  INPUT TIMER VALUE            009
1          DC    H'0'                    RES_FOR_TIAM                 007
1 *
1          @DCEO OTRSUP=,OLINEND=,OMANUAL=,                              C
1                OHCOPY=,OPTAPE=,ONOPOSN=,                               C
1                OHDR=,OETB=,OHOM=,OEXTEND=YES,                          C
1                MODE=LINE,DCEDIT=,OBELL=,OTRANS=,                       C
1                ONOLOGC=,                                               C
1                RDA1=-16,RDA2=-15
2          ORG   *-16
2          DC    AL1(4)
2          ORG   *+16-1
2          ORG   *-15
2          DC    AL1(4)
2          ORG   *+15-1
2                *,@DCEO      999    921011   53531004
1 *
```

```
1          @DCEI DCEDIT=,MODE=LINE,RDA1=-14,RDA2=-13,                       C
1                ITRSUP=,ILINEND=,ICFD=,                                    C
1                IGETBS=,ILCASE=,IHDR=,                                     C
1                IGETFC=,IGETIC=,IEXTEND=YES
2          ORG   *-14
2          DC    AL1(32)
2          ORG   *+14-1
2          ORG   *-13
2          DC    AL1(32)
2          ORG   *+13-1
2                *,@DCEI      999    921011   53531002

1 *
1 S0001S     DS 0Y                                                     A340
1          SVC   39                    SYSFILE SVC
1 *
           CLC   INLNAME(4),='XXXX'
           BNE   QUEST
  END      TERM
1 END      DS    0H                                                     206
1          LA    1,S0006D                                               205
1          B     S0006S                                                 200
1 S0006D     DS 0F                                                      200
1          FHDR  UNIT=6,FUNCT=40,VERS=1                                 207
2          DS    0A
2          DS    0XL8                GENERAL OPERAND LIST HEADER
2          DC    AL2(6)              FUNCTION UNIT NUMBER
2          DC    AL1(40)             FUNCTION NUMBER
2          DC    AL1(1)              FUNCTION INTERFACE VERSION NUMBER
2          DC    X'FFFFFFFF'          Returncode is virgin
1          DC    XL1'01'                                                207
1          DC    XL1'00'
1          DC    XL1'00'
1          DC    XL1'04'
1          DC    CL4'    '
1 S0006S     DS 0Y                                                      200
1          SVC   9
  *
           VTCSET LOG
1 *
1 *              VIRTUAL TERMINAL CONTROL CHARACTER SET
1 *
1 *
1 *              LOGICAL RECORD DELIMITERS
1 *
1 LOGNL    EQU   X'15'           LOGICAL LINE END (CONT NEXT LINE)
1 LOGNP    EQU   X'0C'           LOGICAL PAGE END (CONT NEXT PAGE)
1 LOGCL    EQU   X'0D'           LOGICAL LINE END (CONT SAME LINE)
```

```
1 LOGVPA   EQU   X'29'           LOG VERTICAL POS ABSOLUT (CONT LINE N)
1 LOGHPA   EQU   X'2A'           LOG HORIZONT POS ABSOLUT (CONT COL N)
1 LOGASF   EQU   X'21'           LOG SHEED FEDDING FROM CASETTE N  D1
1 LOGCAP   EQU   X'20'           CONTINUE ACTUAL POSITION AT MSG BEGIN
1 *
1 *              LOGICAL UNIT DELIMITERS
1 *
1 LOGEM1   EQU   X'1D'           EMPHASIZED LAYOUT 1
1 LOGEM2   EQU   X'1F'           EMPHASIZED LAYOUT 2
1 LOGEM3   EQU   X'13'           EMPHASIZED LAYOUT 3
1 LOGEM4   EQU   X'14'           EMPHASIZED LAYOUT 4
1 LOGNOR   EQU   X'1E'           NORMAL LAYOUT
1 LOGDAR   EQU   X'12'           DARK LAYOUT
1 LOGPLD   EQU   X'2B'           PARTIAL LINE DOWN
1 LOGPLU   EQU   X'2C'           PARTIAL LINE UP
1 *
1 LOGSO    EQU   X'0E'           SHIFT OUT TO 2ND CHARACTER SET
1 LOGSI    EQU   X'0F'           SHIFT IN TO NORMAL CHARACTER SET
1 *

1 LOGSPA   EQU   X'36'           START PROTECTED AREA
1 LOGEPA   EQU   X'08'           END PROTECTED AREA
1 LOGNUM   EQU   X'11'           START NUMERIC (UNPROTECTED) AREA
1 *
1 LOGCHS   EQU   X'06'           CHARACTER SET D1D2
1 LOGCOL   EQU   X'17'           COLOUR CHOICE
1 LOGLOC   EQU   X'09'           LOCAL ATTRIBUTE START S1
1 LOGLOX   EQU   X'0A'           LOCAL ATTRIBUTE EXIT S1
1 *
1 LOGVMI   EQU   X'24'           VERTICAL MOVEMENT INDICATOR  D1
1 LOGHMI   EQU   X'23'           HORIZONTAL MOVEMENT INDICATOR  D1
1 LOGLM    EQU   X'38'           LEFT MARGIN  D1D2D3
1 LOGPTS   EQU   X'1A'           PROPORTIONAL TYPING START
1 LOGPTX   EQU   X'1B'           PROPORTIONAL TYPING END
1 LOGMLL   EQU   X'33'           MAXIMAL LINE LENGTH
1 LOGMLN   EQU   X'35'           MAXIMAL LINE NUMBER (ON PAGE)
1 LOGNLQ   EQU   X'39'           NEAR LETTER QUALITY START
1 LOGNLX   EQU   X'3B'           NEAR LETTER QUALITY EXIT
1 *
1 *              SPECIAL FUNCTIONS
1 *
1 LOGDEL   EQU   X'07'           DELETE
1 LOGBS    EQU   X'16'           BACKSPACE
1 LOGSUB   EQU   X'3F'           SUBSTITUTE
1 *
1 *              DELIMITER EXTENSION
1 *
1 LOGEXT   EQU   X'3E'           DELIMITER EXTENSION BYTE
```

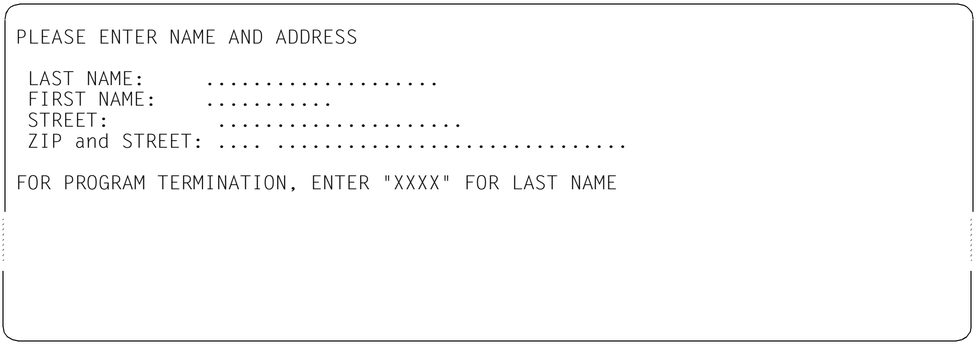
```
1 *
1 *              EXTENDED LOGICAL DELIMITERS
1 *
1 LOGTRA   EQU   C'T'            TRANSPARENT OUTPUT X1L1L2
1 LOGDIM   EQU   C'D'            DIMENSION OF SCREEN D1D2D3D4D5
1 LOGRPT   EQU   C'R'            REPEAT NEXT CHARACTER NN TIMES
1 LOGDIS   EQU   C'I'            SET DISPLAY ATTRIBUTES
1 LOGRS    EQU   X'00'            RESET DISPLAY ATTRIBUTES
1 LOGFL    EQU   X'01'            FLASHING
1 LOGUND   EQU   X'02'            UNDERSCORED
1 LOGBLK   EQU   X'04'            BLANKED
1 LOGRIN   EQU   X'08'            REDUCED INTENSITY
1 LOGINV   EQU   X'10'            INVERSE
1 LOGFLD   EQU   C'F'            SET FIELD CHARACTERISTICS
1 LOGINP   EQU   X'00'            INPUT FIELD
1 LOGPNS   EQU   X'01'            PROTECTED NOT SENDABLE
1 LOGPRS   EQU   X'20'            PROTECTED SENDABLE
1 LOGNUF   EQU   X'02'            NUMERIC
1 LOGMOD   EQU   X'04'            PRE-MODIFIED
1 LOGMAR   EQU   X'08'            MARKABLE
1 LOGPRT   EQU   X'10'            PRINTABLE
1 LOGASK   EQU   X'40'            AUTOMATIC SKIP
1 *
1 *
1 *              PHYSICAL UNIT DELIMITERS
1 *
1 LOGESC   EQU   X'27'           ESCAPE  X

1 LOGDC4   EQU   X'3C'           DC4  X
1 LOGHT    EQU   X'05'           HORIZONTAL TABULATION
1 LOGVT    EQU   X'0B'           VERTICAL TABULATION
1 *
1                *,VTCSET     080    941024   53531028
  INPUT    DS    0CL90
           DS    CL4
  INLNAME  DS    CL20
  INFNAME  DS    CL12
  INSTR    DS    CL30
  INZIP    DS    CL4
  INCITY   DS    CL20
  OUTPUT   DS    0H
           DC    Y(ENDOUT-OUTPUT)
           DS    CL2
           DC    X'01'
           DC    AL1(LOGNP)
           DC    AL1(LOGNL)
           DC    AL1(LOGNL)
           DC    AL1(LOGSPA)
```

```
         DC    AL1(LOGEM3)
         DC    C'PLEASE ENTER NAME AND ADDRESS'
         DC    AL1(LOGNL)
         DC    AL1(LOGNL)
         DC    AL1(LOGSPA)
         DC    C' LAST NAME:     '
         DC    AL1(LOGEPA)
NAME     DS    CL20
         DC    AL1(LOGNL)
         DC    AL1(LOGSPA)
         DC    C' FIRST NAME:    '
         DC    AL1(LOGEPA)
FNAME    DS    CL12
         DC    AL1(LOGNL)
         DC    AL1(LOGSPA)
         DC    C' STREET:        '
         DC    AL1(LOGEPA)
STREET   DS    CL30
         DC    AL1(LOGNL)
         DC    AL1(LOGSPA)
         DC    C' ZIP and CITY:  '
         DC    AL1(LOGEPA)
         DC    AL1(LOGNUM)
ZIP      DS    CL4
         DC    AL1(LOGSPA)
         DC    C'  '
         DC    AL1(LOGEPA)
CITY     DS    CL20
         DC    AL1(LOGNL)
         DC    AL1(LOGNL)
         DC    AL1(LOGSPA)
         DC    AL1(LOGEM3)
         DC    C'FOR PROGRAM TERMINATION, ENTER "XXXX" FOR LAST NAME'
ENDOUT   EQU   *
         END
```

Runtime log:

```
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '<ver>' OF '<date>' LOADED
%  ASS6010 <ver> OF BS2000 ASSEMBH  READY
//compile source=*library-element(macexmp.lib,wrtrd5), -
//        compiler-action=module-generation(module-format=llm), -
//        module-library=macexmp.lib, -
//        listing=parameters(output=*library-element(macexmp.lib,wrtrd5))
%  ASS6011 ASSEMBLY TIME: 314 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 116 MSEC
//end
%  ASS6012 END OF ASSEMBH
/start-executable-program library=macexmp.lib,element-or-symbol=wrtrd5, -
/      prog-mode=*any
%  BLS0523 ELEMENT 'WRTRD5', VERSION '@' FROM LIBRARY
   ':2OSG:$QM212.MACEXMP.LIB' IN PROCESS
%  BLS0524 LLM 'WRTRD5', VERSION ' ' OF '<date> <time>' LOADED
%PLEASE ACKNOWLEDGE
```



```
PLEASE ENTER NAME AND ADDRESS

 LAST NAME:      ....................
 FIRST NAME:     ...........
 STREET:          .....................
 ZIP and STREET: .... ..............................

FOR PROGRAM TERMINATION, ENTER "XXXX" FOR LAST NAME
```

6 Appendix

The appendix contains the following sections and tables:

- macros supported only for compatibility

  CDUMP
  GETSW
  GETUS
  HSITYPE
  MRSINF
  MRSSTA
  MSG7
  SETSW
  SETUS
  SINF
  TABLE

- a table containing all macros described in the manual in alphabetical order
- a table containing all macros described in the manual by SVC numbers
- a table with other macros of BS2000 OSD/BC not described in this manual.
- a table with the standardized function key codes

6.1 Macros supported only for compatibility

CDUMP – Output user, system or area dump

General

Application area:	Debugging aids; see page 162
Macro type:	for user and system dumps only: Type S, MF format **1**: standard E/L/D form; for area dumps only: Type S, MF format **2**: standard E/L/D/C form; see page 29

- The new **CDUMP2** macro is available as of BS2000/OSD-BC V3.0.

Macro description

The **CDUMP** generates a dump (in its own dump task) for the task which has called **CDUMP**. By specifying the SCOPE operand, the user can determine whether an area dump, a user dump or a system dump is to be output.

Macro formats and description of operands

In the following format diagram, separate call formats are specified for generating a system dump, user dump or area dump.

Output of an area dump

CDUMP
SCOPE=AREA, { NUM=#n AREAS=((start1,end1),(start2,end2),...) } ,PC=STD / addr [,TITLE=addr] ,MODE=STD / **EXP**ANDED [,DS@=addr] ,MF=S / E / L / C / D [,PARAM=addr / (r)] ,ID=CD / pre

Output of a system dump

CDUMP
SCOPE=SYSTEM ,PC=STD / addr / (r) ,EC=STD / addr / (r) ,IW=STD / PC [, { CODE= { 'name' addr (r) } INSERT= { addr (r) } }] [,TITLE=addr / (r)] ,DIAG=NO / YES [,ELSN=addr / (r)] ,PARMOD=24 / 31 ,MF=S / L / (E,..) / D ,ID=CD / pre

Output of a user dump

```
CDUMP

[SCOPE=USER]
,PC=STD / addr / (r)
,EC=STD / addr / (r)
,IW=STD / PC
     ⎧         ⎧ ’name’ ⎫  ⎫
     ⎪ CODE=   ⎨  addr  ⎬  ⎪
[,   ⎨         ⎩  (r)   ⎭  ⎬  ]
     ⎪                     ⎪
     ⎪ INSERT= ⎧  addr  ⎫  ⎪
     ⎩         ⎩  (r)   ⎭  ⎭

[,TITLE=addr / (r)]
,DS=STD / NO / YES / listaddr
,DIV=STD / NO / YES
,PARMOD=24 / 31
,MF=S / L / (E,..) / D
,ID=CD / pre
```

The operands are described in alphabetical order below.

AREAS=
Defines the areas to be included in the dump by means of their start and end addresses. These may be specified in any form permissible for an address constant (see example). A list of up to 4 memory areas may be specified. AREAS is prohibited if MF=C/D is specified.

((start1,end1),....)
start1 = address of the first byte (start address) of the memory area to be dumped.
end1 = address of the last byte (end address) of the memory area to be dumped.

CODE=
Gives a string which identifies the dump. This character string is output with message `IDA0N51`. This operand is permitted only in conjunction with SCOPE=USER/SYSTEM.

addr
Symbolic address of the field with any character string;
length = 7 bytes.

(r)
Register containing the address value “addr”.

'name'
name = any character string; length = 7 characters.

DIAG=
Determines whether or not a message indicating the address of the CDUMP-SVC is sent to the operator. This operand is permitted only in conjunction with SCOPE=SYSTEM.

NO
Default setting: no message is output.

YES
The message is output.

DIV=
Specifies whether DIV windows are to be included in the user dump. This operand may be specified only in conjunction with SCOPE=USER and is not evaluated unless PARMOD=31.
If PARMOD≠31 the value DIV=YES is used irrespective of the parameter specification. See also the section “Extended addressing with data spaces” on page 61.

STD
Default setting: the value set in the MODIFY-TEST-OPTIONS command determines whether DIV windows are to be included in the user dump (YES) or not (NO).

NO
No DIV windows are to be included in the user dump.

YES
All DIV windows are to be included in the user dump.

DS=
Determines which data spaces (DS) are to be included in the user dump. This operand may be specified only in conjunction with SCOPE=USER and is not evaluated unless PARMOD=31. If PARMOD≠31 the value DS=YES is used irrespective of the parameter specification. See also the section “Extended addressing with data spaces” on page 61.

STD
Default setting: the value set in the MODIFY-TEST-OPTIONS command determines whether data spaces are to be included in the user dump (YES) or not (NO).

listaddr
Address of a list of SPIDs (8 bytes per entry). The list must end with a zero entry (D(0)).

NO
No data spaces are to be included in the user dump.

YES
All data spaces (up to 100 data spaces used by the caller) are to be included in the user dump.

DS@=
Points to a data space control block (DSCB) whose DSECT is generated with MF=D. A data space and its associated areas can be defined in the DSCB. This operand may be specified only in conjunction with SCOPE=AREA.

Data structure of a DSCB:

```
CDDDSCB   DSECT ,      DATA SPACE CTRL BLOCK
CDDDS@    DS    A                Points to the next DSCB
CDDSPID   DS    XL8              SPID of the DS
CDDNUM    DS    H                Number of areas in the DS
CDDSTRT   DS    A                Start of the first area
CDDEND    DS    A                End of the first area
```

By chaining DSCBs, users can specify areas of more than one data space of their task. By default, no areas of a data space are specified.
See also the section “Extended addressing with data spaces” on page 61.

addr
Symbolic address of the DSCB.
addr = NULL is the default value.

EC=
Defines the location from which the event code (interrupt weight) is to be fetched. This specification is permissible only in conjunction with SCOPE=USER/SYSTEM; for SCOPE=AREA the event code is always fetched from the calling stack.

STD
Default setting: the event code is to be fetched from the calling stack.

addr
Symbolic address of the field containing the event code.

(r)
r = Register containing the event code (right-justified)

ELSN=
Specifies the address of a field containing the number of an Error Log Sequence Block to which specific data of the error logging file has been written by the caller.
Field length = 4 bytes, which must be aligned on a word boundary. The number must be entered as a binary digit. This operand is only permitted in conjunction with SCOPE=SYSTEM.

addr
Symbolic address (name) of the field

(r)
r = Register containing the address value “addr”

INSERT=
Defines a text to be output with the message `IDA0N51`. This text could contain more detailed information on the cause of the dump.
The user must place this text, which may be up to 60 characters long, in a data area with the following format:

Byte 1: Length (hexadecimal) of the text to be output (in bytes). If byte 1 has the value 0, no INSERT text is output.

Byte 2 through n (n≤61): Text to be output.

The INSERT operand is ignored when SCOPE=AREA.

addr
Symbolic address of the data area which contains length specification and text.

(r)
r= Register with the address of the data area which contains length specification and text

IW=
Defines the location from which the interrupt weight is to be fetched. This specification is permissible only in conjunction with SCOPE=USER and only for the 24-bit interface. The operand is supported for compatibility reasons only; EC=... should be used in programs instead.

<u>STD</u>
Default setting: specifies that the interrupt weight is to be fetched from the calling stack.

PC
Specifies that the interrupt weight is contained in byte 1 of the field/register specified with the PC operand.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A prefix ID (pre = 2 letters) can be specified for MF=C or MF=D, as shown in the macro format.

The following operands must be specified for area dumps: the SCOPE operand if MF=C/D/E and additionally the NUM operand if MF=C/D.
For system dumps, the SCOPE operand is mandatory if MF=C/D/E.

MODE=
Defines the scope of the diagnostic data to be output for an area dump. The operand may be specified only in conjunction with SCOPE=AREA.

<u>STD</u>
Default setting: ensures that only the AIDSYSD module and the areas with COMAREA, P1-PCB and TCB are dumped in addition to the specified areas in class 6 and class 5 memory (see the section describing macros and area dumps).

EXPANDED
Initiates dumping of the area with COMAREA and all other system areas as in user dump in addition to the areas specified in class 6 and class 5 memory (see the section describing macros and user dumps).

NUM=
Gives the number of areas to be dumped. NUM is permissible only in conjunction with MF=L/C/D. The start and end addresses of the areas to be dumped must be entered in the generated operand list (dynamically). If MF=C/D, NUM is mandatory.

#n
n = Number of areas to be dumped; 1 ≤ n ≤ 2048.
(The '#' character must precede the number).

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit-interface is generated.

If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro.

24
This specification is not permissible in conjunction with SCOPE=AREA. The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

PC=
Defines a register or field containing the program counter to be logged.

<u>STD</u>
Default setting: the program counter is to be fetched from the calling stack.

addr
Symbolic address (name) of the field (word) containing the program counter.

(r)
r = Register containing the program counter. This specification is permissible only in conjunction with SCOPE=USER/SYSTEM.

SCOPE=
Defines whether a user, system or area dump is to be generated.

USER
Default setting: a user dump is generated.

SYSTEM
A system dump is generated. This specification may be entered by users with a read privilege ≥ 3 only.

AREA
An area dump is generated.

TITLE=
Defines a second title line for the memory dump
(length = 132 characters).

Note

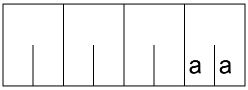
This operand is still supported for reasons of compatibility only, i.e. DAMP does not process this title line. Specifying this operand therefore has no effect.

addr
Symbolic address of the field containing the title line.

(r)
r = Register containing the address value “addr”. This specification is permissible only in conjunction with. SCOPE=USER/SYSTEM

Return information and error flags

R15 /
standar d
header:



A return code relating to the execution of the CDUMP macro is transferred in the rigtmost byte of register R15. When using the 31-bit-interface, the return code is additionally transferred in the standard header (main code).

X'aa'	**Meaning**
X'00'	Dump completed without errors.
X'04'	Dump completed using default values.
X'08'	Dump suppressed due to /OPTION DUMP operand.
X'0C'	Dump suppressed due to systems standards.
X'10'	Dump suppressed due to severe CDUMP operand error.
X'14'	Dump suppressed due to insufficient test privilege.
X'18'	Dump suppressed due to error in DMS routines.
X'1C'	Dump suppressed due to system error.
X'20'	Dump suppressed due to previous interruption of CDUMP.
X'24'	Dump suppressed due to foreign task dump error.
X'28'	Dump suppressed due to shutdown processing.
X'2C'	Dump suppressed by calling task.
X'30'	All area specifications are invalid. No dump is output.
X'34'	Incorrect specification for NUM=.... No dump is output.
X'38'	Some dump areas are not located in the caller' s address space or the specifications are inconsistent. At least one memory area has, however, been output.
X'3C'	Dump suppressed since “DMS READY” has not yet been reached.

GETSW – Get switch

General

Application area:	User and job switches; see page 29
Macro type:	Type O; see page 28

- As of BS2000/OSD-BC V1.0, the new macro **SWITCH** is available. This macro combines the functionality of the GETSW, GETUS, SETSW and SETUS macros.

Macro description

(For a general description of the job switch see the **SETSW** macro)

The macro **GETSW** copies into register R0 the settings of the job switches of the job that is currently running.
The switches are assigned to the bits of register R0 in ascending sequence (from right to left):

bit 2^0 → switch 0

bit 2^1 → switch 1

: :

: :

bit 2^{31} → switch 31

where: bit $2^n = 0$: switch n off

bit $2^n = 1$: switch n on

$0 \leq n \leq 31$

There is no return code to report on execution of the macro.

Macro format

GETSW

GETUS – Get user switch

General

Application area:	User and job switches; see page 29
Macro type:	Type O; see page 28

- As of BS2000/OSD-BC V1.0, the new macro **SWITCH** is available. This macro combines the functionality of the GETSW, GETUS, SETSW and SETUS macros.

Macro description

For a general description of the user switch see the **SETUS** macro.

The macro **GETUS** copies the settings of the user switches for the specified user ID into register R0. Users can specify their own user ID or that of another user.
The switches are assigned to the bits of register R0 in ascending sequence (from right to left):

bit 2^0 → switch 0
bit 2^1 → switch 1
: :
: :
bit 2^{31} → switch 31

where: bit 2^n = 0: switch n off
bit 2^n = 1: switch n on
$0 \leq n \leq 31$

Macro format and description of operands

GETUS
[(1)]

(1)
Register R1. Register R1 must contain the address of a field containing the user ID.
The user ID must be copied left-justified into the address field (specification without “$”; to be padded with blanks to eight positions, if required).

Macro call without operand
The switch settings for the user ID specified in the SET-LOGON-PARAMETERS command will be supplied.

Return information and error flags

R15:	0	0	0	0	0	0	a	a

A return code relating to the execution of the GETUS macro is transferred in the rigtmost byte of register R15.

X'aa'	Meaning
X'00'	Normal execution.
X'04'	Operand error.
X'08'	The user ID does not exist.
X'0C'	The user ID is no longer valid.
X'20'	Internal error.

HSITYPE – Output information about current HSI

General

Application areas:	Requesting and accessing lists and tables; see page 155 XS programming; see page 164
Macro type:	Type R; see page 28

- HSITYPE is replaced by the **NSIINF** macro.

Macro description

The **HSITYPE** macro provides the user with information about the current HSI (Hardware-Software Interface). The information is output to a field whose address is stored in a register. The information provided applies to the entire duration of the current BS2000 session.

Only output value (only XS31 hardware is supported):

XS31: Addressable memory area > 16 Mb; all addresses are interpreted as either 24-bit or 31-bit addresses, depending on the addressing mode.

The macro generates a 31-bit interface.

Macro format and description of operands

HSITYPE
(r)

(r)
Register containing the address of the field where the information is to be entered. Field length = 4 bytes; the field must be aligned on a word boundary.

Register contents

- Register R1 is overwritten during macro execution.
- Register R15 contains the return code.

Return information and error flags

R15:

0	0	0	0	0	0	a	a

A return code relating to the execution of the HSITYPE macro is transferred in register R15.

X'aa'	Meaning
X'00'	Function executed.
X'04'	Function not executed. Invalid address was specified or incorrect register.
X'08'	Function not executed. System error (in the event of internal REQM call).
X'18'	Function not executed. Invalid macro format.

MRSINF – Request MSCF information

General

Application area:	Multiprocessor systems; see page 164
Macro type:	Type S, MSCF format **2**: standard/C/D/L/E/M form; see page 29

- The new macro **MCSINFO** is available as of BS2000/OSD-BC V3.0 and replaces the MRSINF and MRSSTA macros.

Macro description

The **MRSINF** macro gives the user information on selected processors or on all processors in an MSCF communication network. A record with the following information is placed in an output area to be made available by the user:

– BCAM name of the processor

– External representation of the SYSID processor

– BS2000 version which is used with this processor

– Attainability of the processor in the MSCF network: a distinction is made between a local processor, a processor with which a connection exists, and a processor with which no connection exists.

– Type of processor network: a distinction is made between a processor defined for LCS (Loosely Coupled System) and one defined for CCS (Closely Coupled System).

Macro format and description of operands

MRSINF
KEY=HOST / SYSID / ALL [,HOST='bcamname' / addr] [,ESYSID='sysid' / addr] [,AREA=addr / (r)] [,AREAL=length / addr] ,MF=S / E / L / M / C / D [,PARAM=addr / (r)] ,PREFIX=M / p ,MACID=RSI / macid ,VERSION=2 / 1

KEY=
Indicates whether information is to be output about the entire MSCF network or about a specific processor and specifies the operands for selecting the processors.

HOST
Default value; information is requested about the processor in the MSCF communication network whose BCAM name is specified in the HOST operand.

SYSID
Information is requested about the processor in the MSCF communication network whose SYSID (system identification) is specified in the ESYSID operand.

ALL
Information about the entire MSCF communication network is requested.

HOST=
Specifies with the BCAM name the processor about which information is output.

'bcamname'
BCAM name of the processor about which information is requested. The name must be enclosed in apostrophes.
This operand value is to be specified with MF=S or MF=L.

addr
Symbolic address of a field in which the user places the BCAM name of the processor.
This operand value can be specified with MF=M.

ESYSID=
Specifies with a SYSID the processor about which information is output.

'sysid'
SYSID of the processor about which information is requested. sysid may be one to three characters long and must be enclosed in apostrophes.
This operand value is to be specified with MF=S or MF=L.

addr
Symbolic address of a field in which the user places the SYSID of the processor.
This operand value can be specified with MF=M.

AREA=
Specifies the address of an output area in which **MRSINF** transfers the requested information.

addr
Symbolic address of a field for accepting the requested MCSF information. This field is to be aligned on a word boundary.
With KEY=ALL, up to 164 output records each 16 bytes in length can be written to this field. The first output record always contains the information about the local processor.

(r)
r =Register containing the address value “addr”. A register can be specified only with MF=M.

AREAL
Specifies the length of the output area for the requested MSCF information.
If the output area selected is too small the output information is truncated. The user is informed of this with a return code.

length
Length (in bytes) of the output area, i.e. the field containing the address “addr”.
This operand value is to be specified with MF=S or MF=L.

addr
Symbolic address of a field (halfword) in which the user places the value for the length (as a binary number).
This operand value can be specified with MF=M.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “macro type” and are included in the macro format.

A PREFIX can be specified in the C form, D form or M form of the macro and additionally a MACID in the C form or M form (see section “S-type macros” on page 29).

Return information and error flags

standar d header:

0	0	b	b	a	a	a	a

The following return code relating to execution of the MRSINF macro is transferred in the standard header (bb=Subcode1,aaa=Maincode).

X'bb'	X'aaaa'	Meaning
X'00'	X'0000'	Function executed successfully; no error.
X'01'	X'0001'	Operand values not in the permissible value range.
X'01'	X'0002'	The output area is too short. Information was truncated.
X'01'	X'0003'	The output area is not aligned on a word boundary.
X'01'	X'0004'	An invalid address was output for the output area.
X'20'	X'0020'	Internal error. The SERSLOG file contains further details on the cause of error.
X'40'	X'0040'	The specified processor is not known.
X'40'	X'0041'	The specified SYSID is not known.
X'40'	X'0042'	The specified SYSID is invalid.

Other return codes which, in accordance with conventions, apply to all macros are given in the table “Standard header” on page 43.

The calling program is terminated when the following errors occur:

- The data area is not assigned to the caller.
- The data area is not aligned on a word boundary.
- The data area is protected against write access.

MRSSTA – Display MSCF status

General

Application area: Multiprocessor systems; see page 164
Macro type: Type S, MF format **1**: standard/L/E form; see page 29

- The new macro **MCSINFO** is available as of BS2000/OSD-BC V3.0 and replaces the MRSINF and MRSSTA macros.

Macro description

This macro is only available to users of the multiprocessor system (see the “HIPLEX MSCF” manual [26]).

The **MRSSTA** macro gives information on all active and potential connections between the local processor and other processors in the MSCF network.

Macro format and description of operands

MRSSTA

`{ HOST= { 'bcamname' / addr1 / (r1) } }`
`{ AREA= { addr2 / (r2) } }`
 	
,MF=<u>S</u> / (E,..) / L

```
 ⎧        ⎧ ’bcamname’ ⎫ ⎫
 ⎪  HOST= ⎨ addr1      ⎬ ⎪
 ⎪        ⎩ (r1)       ⎭ ⎪
 ⎨                       ⎬
 ⎪  AREA= ⎧ addr2 ⎫      ⎪
 ⎩        ⎩ (r2)  ⎭      ⎭

,MF=S / (E,..) / L
```

HOST=
Specifies a host computer whose connection to the local processor is to be interrogated. If this operand is specified, an appropriate return code will be set in register R15 only.

bcamname
Is the processor's BCAM name (as defined during BCAM generation).

addr1
Symbolic address of a BCAM name.

(r1)
r1 = Register that contains the address of the BCAM name.

AREA=
Specifies the address of an area in which all active and potential connections to the local processor are to be stored. The area must start on a halfword boundary. Before a macro is called, the length of the user area must be entered in the first two bytes (length field). After the macro is executed the length field will contain the total length of all entries for the processor name. The first name is always the name of the local processor. If the HOST operand is also specified, the AREA operand will be ignored.

After execution of the macro, the area specified by AREA contains the following information:

Byte 1 - 8:	Processor name
Byte 9:	Return code from register R15, which is set when the HOST operand is specified (except X'04A7')
Byte 10:	X'FF' in the last entry, otherwise X'00'

addr2
Symbolic address of the area in which the connections are to be stored.

(r2)
r2 = register which contains the address of the area

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

Notes

Since the length of a processor name entry is 10 bytes, the required size of the user area is calculated as number of processors * 10 + 2 bytes.

See also the “HIPLEX MSCF” manual [26] for information on MSCF communication.

Return information and error flags

R15:

0	0	0	0	0	0	a	a

A return code relating to the execution of the MRSSTA macro is transferred in the rigtmost byte of register R15.

Return code	Meaning
X'00'	HOST is local.
X'00'	All known processor names are stored in the user area.
X'04'	HOST is connected to the local MSCF.
X'08'	HOST is not connected to the local processor.
X'0C'	MSCF communication is not active.
X'10'	Processor unknown in MSCF network.
X'14'	Operand error.
X'18'	Specified length too small to store all names.
X'FF'	MSCF is not included in this BS2000.
X'04A7'	Invalid address of the MRSSTA operand list.

MSG7 – Output message

General

Application area: Messages; see page 161
Makrotyp: Type S, MF format **1**: 31-bit interface: standard/L/D/C/E form; see page 29

- The **MSG7** macro will not be developed further; new functions will only be offered in the **MSG7X** macro which, unlike the **MSG7** macro, also uses the new data area layout.

Macro description

The **MSG7** macro outputs a system message to SYSOUT, SYSLST, the operator console or a user program area. The specified message code must contain 7 characters.
The operand values number2, A, B, C, N, R, T serve to structure a CSECT/DSECT and can be specified only in conjunction with the MF=C/D form of the macro.

Every system message has a 7-character message code. The first 3 characters of the code denote the message class; the remaining 4 characters are used for consecutive numbering within a class. System messages may contain variable sections “(&nnn)” that can be replaced by inserts.

Macro format and description of operands

```
MSG7
---------------------------------------------------------------
ID=msgid / (r1) / ((r1),(r2)) / (class,(r)) / R / C

            ⎧ (insertlength, ⎧ addr                 ⎫)          ⎫
            ⎪                ⎩ base,[index],[displ] ⎭           ⎪
            ⎪                                                   ⎪
            ⎪ ((insertlength, ⎧ addr                 ⎫),...)    ⎪
[,INSERT=   ⎨                 ⎩ base,[index],[displ] ⎭          ⎬]
            ⎪ number1                                           ⎪
            ⎪ number2                                           ⎪
            ⎪ (number,B)                                        ⎪
            ⎩ (number,T)                                        ⎭
[,LAN='language']

,DEST=SYSOUT / SYSLST / CONSOLE / (destination,...)

[,UCDEST='destcode' / (r) / destaddr / N / R / A]
```

MSG7 (cont.)

```
           ⎧ (replylength,⎰ addr                 ⎱)  ⎫
           ⎪              ⎱ base,[index],[displ] ⎰   ⎪
           ⎪ REG                                     ⎪
[,REPLY=   ⎨ N                                       ⎬]
           ⎪ A                                       ⎪
           ⎪ B                                       ⎪
           ⎩ R                                       ⎭

            ⎧ (bufferlength,⎰ addr                 ⎱)  ⎫
            ⎪               ⎱ base,[index],[displ] ⎰   ⎪
[,BUFFER=   ⎨ N                                        ⎬] [,MAP=NO / YES]
            ⎪ A                                        ⎪
            ⎩ B                                        ⎭

,DMS=APPL / NOTAPPL / NA

[,RC=X / (r) / R / N]

,MF=S / C / (C,pre) / (E,...) / (D,pre) / D / L
```

ID=
Specifies the message code of the system message to be output.

msgid
Specifies a 7-character message code.

(r1)
Register pair r1 and r1 + 1 containing the message code:
r1 = Register containing the message class (right-justified).
r1 + 1 = Register containing the message number.
r1 must be even-numbered.

((r1),(r2))
Register pair containing the message code:
r1 = Register containing the message class (right-justified).
r2 = Register containing the message number.

(class,(r))
class = 3-character message class
r = register containing the 4-character message number.

INSERT=

Specifies up to 15 lengths and addresses of inserts. An address reference is created in the message processing operand list for each entry.
If more inserts are specified than can be accommodated by the message, excess inserts are ignored. An insert consisting solely of blanks is shortened to a single blank.
Trailing blanks in an insert are suppressed.
The character X'01' can be entered at the end of an insert to prevent the suppression of blanks. When the character X'01' is encountered during message processing, all blanks that precede it are retained.
If a message text contains more inserts than are specified in the macro call, the default value is assumed for every excess insert. The substitute insert (&nn) is used if no default value is present.

(insertlength,...)

`insertlength` = length of the insert
`addr` = symbolic address (name) of the area
`base` = base register [1)]
`index` = index register [1)]
`displ` = displacement [1)]

[1)] Entries for calculating the address of the insert

Inserts may be skipped by entering commas for omitted positions e.g. INSERT=(,(insertlength2,addr2),,(insertlength4,addr4)). Inserts omitted in this way are replaced by their default values or by the substitute insert (&nn).

If insertlength=0, the insert must begin with a record length field (4 bytes):
Bytes 0-1: length of the insert
Bytes 2-3: reserved.

Notes

- The commas should always be entered in the case of indexed addressing.
- An insertion list begins with two parentheses.
- The sum of the insertion lengths ≤ 4079 bytes.

number1

Number of INSERTs for which space is to be reserved in the operand list for message processing.
“number1” may be specified only in conjunction with MF=L.

number2

Number of inserts for which names are to be generated in the CSECT/DSECT.
“number2” may be specified only in conjunction with the C form or the D form of the macro.

(number,B)
Number of inserts whose addresses are to be specified in the form (base,index,displ).
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

(number,T)
Number of inserts for which symbolic addresses are to be specified.
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

LAN=
Specifies the language to be used for message output.
This operand is ignored if DEST=CONSOLE is specified.

'language'
1 letter to identify the language:
D = German, E = English.
For additional options please consult system administration. The default value is defined by the system parameter MSGLPRI (see the “Introduction to System Administration” manual [10]; this value is also assumed in the event of invalid specifications.

REPLY=
Defines a reply area. The area must be aligned on a halfword boundary and begin with a record length field (4 bytes: bytes 1-2: length of the reply; bytes 3-4 reserved). Before the macro is executed, bytes 1-2 must contain the length of the reply area (≤ 4095 bytes). When the macro is executed, the current length of the reply is entered in bytes 1-2.

Lowercase letters are converted to uppercase when entered via the REPLY operand. REPLY may be specified only in conjunction with DEST=SYSOUT/CONSOLE.

If a “?” reserved in MIP as a keyword is entered as a reply, MIP MIP shows the meaning and action of the relevant message before the message is again output for reply.

Note

REPLY may be specified only in conjunction with DEST=SYSOUT/CONSOLE.

(replylength,...)

`replylength`	= length of the reply area > 4 bytes
`addr2`	= symbolic address (name) of the area
`base`	= base register [1)]
`index`	= index register [1)]
`displ`	= displacement [1)]

[1)] Entries for calculating the address of the area

Note

The commas should <u>always</u> be entered in the case of indexed addressing.

REG
Specifies that the reply is written left-justified in register R1. Reply length ≤ 4 bytes.

N
No reply area is generated in the CSECT/DSECT
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

A
The address of the reply area is a symbolic address.
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

B
The address of the reply area is given in the form (base, index,displ).
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

R
The reply is to be transferred in register R1.
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

DEST=
Designates destinations for the converted system message. If the REPLY operand is also specified, only DEST=SYSOUT/CONSOLE is permissible.

<u>SYSOUT</u>
Output to SYSOUT.

SYSLST
Output to SYSLST.

CONSOLE
Output to the operator console.

(...)
Combination of the above-mentioned destinations; entry in parentheses.

BUFFER=
Specifies an area to which the converted system message is to be transferred. BUFFER must be aligned on a halfword boundary.
A record length field is entered in the first 2 bytes:

Bytes 0-1: length of the area
Bytes 2-3: reserved
Bytes 4: output control character
Bytes 5-n: message text

} WROUT format

(bufferlength,...)

`bufferlength`	= length of the area > 16 bytes
`addr`	= symbolic address (name) of the area
`base`	= base register [1)]
`index`	= index register [1)]
`displ`	= displacement [1)]

[1)] Entries for calculating the address of the area

N
No area for the converted system message is generated in the CSECT/DSECT.
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

A
The address of the area is a symbolic address.
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

B
The address of the area is given in the form (base, index,displ).
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

MAP=
Specifies whether BUFFER is given another structure (mapping format).

NO
BUFFER is created in the **WROUT** format.

YES
BUFFER is created in the mapping format (see below).

UCDEST=
UCON destination (UCON=Universal CONsole); (see “Introduction to System Administration” [10] manual).
The destination of a message can be specified as follows:

- mnemonic device name for a particular console.
- routing code for consoles and authorized user tasks which are allocated a particular area of activity.
- authorization name for an authorized user task.

Notes

- UCDEST is not executed unless DEST=CONSOLE was specified.
- UCDEST has priority over the RC operand.

'destcod'
The following entries are permissible for 'destcode':

– '(mn)'
  where mn = 2-character mnemonic device name.
– '< x'
  where x = routing code; < must be specified.
– 'name'
  where name = name of the user task (4 characters).

destaddr
Symbolic name of an area (word length) with the entry for “destcode”; left-justified entries; alignment on a word boundary.

(r)
r = Register containing the entries for “destcode”; left-justified entry.

N
No field for “destcode” is to be generated in the CSECT/DSECT.
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

R
The entry for 'destcode' is entered in a register.
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

A
The address of the field containing “destcode” is a symbolic address (destaddr).
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

DMS=
Specifies the message search mechanism.

<u>APPL</u>
The message is sought with the aid of DMS (message files and DLAM area).

NOTAPPL / NA
The message is sought at system level only in the DLAM area without DMS). Reply if the requested message is not found:
`msgid,DMS NOT IN MEMORY,program counter,modul,inserts.`

RC=
Specifies the routing code for sending a message to the console. The routing code entered in the message unit is ignored if the RC operand is specified.

x
Routing code (1st character). See also the “Introduction to System Administration” manual [10].

(r)
Register containing the specification for “x”; right-justified entry.

R
The routing code is entered in a register.
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

N
No field for the routing code is generated in the CSECT/DSECT.
This specification is required for the structuring of the CSECT/DSECT; it may be specified only in conjunction with the C form or the D form of the macro.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

A prefix (pre = 1..4 letters) can be specified in the C form and D form, as shown in the macro format.

Default values:	pre = MSGC for the C form
	pre = MSGD for the D form

If less than 4 letters are entered for the prefix, the string MSGx (where x ≙ 1st letter) results.

The operand list also contains a definition of the output area, reply area and fields for a maximum of 15 inserts.

Mapping format

The mapping format specifies the structure of the BUFFER area in the case of MAP=YES. Various entries (mapping list) are prefixed to the structure defined under BUFFER.

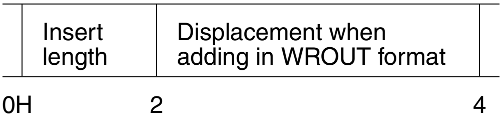
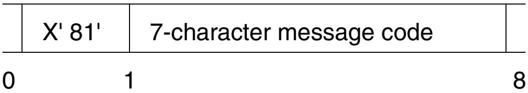
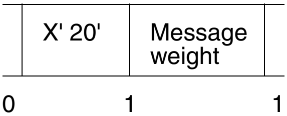
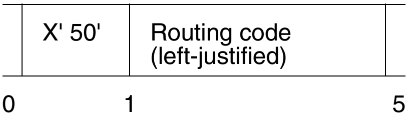
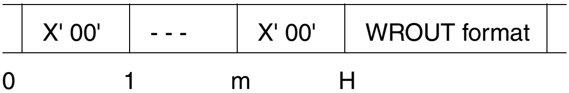
Structure of the mapping format:

Bytes 0-1:	length of the mapping list
Bytes 2-3:	C'MP'
Bytes 4-n:	entries
Bytes n+1 bis p:	remainder of structure as under BUFFER (WROUT format).

The entries contain information on

- the inserts
- the message code
- the routing code
- the message weight
- slack bytes for alignment.

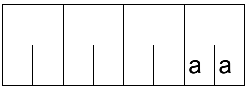
Entries are stored as follows:

Entry on	Structure of the entry
Insert	 Insert length Displacement when adding in WROUT format 0H 2 4 (A separate entry ... is created for each insert)
Message code	
Message weight	
Routing code	
Slack bytes	 X' 00' - - - X' 00' WROUT format 0 1 m H

Return information and error flags

The operand list also contains a definition of the output area, reply area and fields for a maximum of 15 inserts.

R15:



A return code relating to the execution of the MSG7 macro is transferred in the rightmost byte of register R15.

X'aa'	**Meaning**
X'00'	MSG7 macro completed without errors
X'04'	Abnormal termination of input/output
X'08'	Invalid message number, area address, mnemonic device name, name of the authorized user task or invalid area length
X'0C'	Noninteractive user requests reply
X'10'	No memory space available
X'14'	Break during WROUT macro
X'18'	Truncated message written to output area
X'20'	Message output aborted
X'2C'	DEST=SYSLST was specified with REPLY=...
X'30'	DEST=(...) was specified with REPLY=...
X'41'	The MIP subsystem is not loaded

SETSW – Set job switch

General

Application area: User and job switches; see page 73
Macro type: Type R; see page 28

- As of BS2000/OSD-BC V1.0, the new macro **SWITCH** is available. This macro combines the functionality of the GETSW, GETUS, SETSW and SETUS macros.

Macro description

Users can set any of the switches assigned to their job on or off, or invert them, using the **SETSW** macro.

Macro format and description of operands

SETSW

[{ ON=(no,...,no)
OFF=(no,...,no)
INVERT=(no,...,no) }]

ON=(...)
The switches specified are set to “on”.

OFF=(...)
The switches specified are set to “off”.

INVERT=(...)
The switches specified are set to “on” if they were “off” and to “off” if they were “on”.

no
Number of a job switch which is to be changed. The job switches are numbered 0 through 31.
Several switches may be specified in any order or as a consecutive series of switches (e.g. 3-8).

Calling the macro without operands:

Instead of specifying the operands, the user may specify the desired settings of all job switches in register R0. The job switches correspond to the bits of the register, reading from right to left.

bit 2^{0} → switch 0
bit 2^{1} → switch 1
   :   :
   :   :
bit 2^{31} → switch 31

where: bit 2^{n} = 0: switch n off
bit 2^{n} = 1: switch n on
$0 \leq n \leq 31$

Return information and error flags

R15:

0	0	0	0	0	0	a	a

A return code relating to the execution of the SETSW macro is transferred in the rigtmost byte of register R15.

X'aa'	Meaning
X'00'	The switch setting was changed as desired.
X'04'	Operand error. The switch setting was not executed.
X'20'	Internal error.

SETUS – Set user switch

General

Application area: User and job switches; see page 73
Macro type: Type R; see page 28

- As of BS2000/OSD-BC V1.0, the new macro **SWITCH** is available. This macro combines the functionality of the GETSW, GETUS, SETSW and SETUS macros.

Macro description

Users can set any of the user switches assigned to their user ID on or off, or invert them, using the **SETUS** macro.

Macro format and description of operands

SETUS

[{ ON=(no,...,no)
OFF=(no,...,no)
INVERT=(no,...,no) }]

no
Number of a user switch which is to be changed. The user switches are numbered 0 through 31.
Several switches may be specified in any order or as a consecutive series of switches (e.g. 3-8).

ON=(...)
The switches specified are set to “on”.

OFF=(...)
The switches specified are set to “off”.

INVERT=(...)
The switches specified are set to “on” if they were “off” and to “off” if they were “on.

Calling the macro without operands:

Instead of specifying the operands, the user may specify the desired settings of all user switches in register R0. The user switches correspond to the bits of the register, reading from right to left.

bit 2^0 → switch 0
bit 2^1 → switch 1
: :
: :
bit 2^{31} → switch 31

where: bit $2^n = 0$: switch n off
bit $2^n = 1$: switch n on
$0 \leq n \leq 31$

Return information and error flags

R15:

0	0	0	0	0	0	a	a

A return code relating to the execution of the SETUS macro is transferred in the rigtmost byte of register R15.

X'aa'	Meaning
X'00'	The switch setting was changed as desired.
X'04'	a) Operand error. The switch setting was not executed. b) The switch settings affect one of the user' s jobs which is in the WHEN queue. The specified switch settings have been executed.
X'20'	Internal error.

SINF – Output system information

General

Application area: Requesting and accessing lists and tables; see page 155
Macro type: Type S, MF format **1**: standard/E/L form; see page 29

As of BS2000/OSD-BC V1.0, two new macros **NSIINF** and **NSIOPT** are available. These macros incorporate the full functionality of the **SINF** macro. The functionality of the **SINF** macro has been frozen as of BS2000 V10.0. The macros **NSIINF** and **NSIOPT** should be used for new developments.

Macro description

The **SINF** macro provides information about the following:

CPU: serial numbers and identifications of available CPUs
BS2000: label and version of the operating system, as well as addressing mode of the operating system
Memory: size of the (physical) main memory and start address of the operating system in virtual address space
HSI: HSI base type
Server: type of server (model series)
System parameters:
system parameters of significance for the nonprivileged user such as DEFLUID or TEMPFILE

The selected information is transferred to a specified field. No more than one item of information can be requested per macro call (see the list at the end of the description of operands).

Macro format and description of operands

SINF
INFO='info' / addr / (r) ,FIELD=addr / (r) ,LENGTH=length / (r) [,PARMOD=24 / 31] [,MF=(E,..) / L]

INFO=
Specifies the type of information to be output.

'info'
Identifier for the item of information. Please refer to the list at the end of the description of operands for possible entries.

addr
Symbolic address of an 8-byte field containing the value “info” (left-justified, with trailing blanks).

(r)
Register containing the address value “addr”.

FIELD=
Specifies the symbolic address of the field to which the requested information is output.
Field length ≥ length of information (cf. list). The information is entered left-justified.

addr
Symbolic address of the field.

(r)
Register containing the address value “addr”.

LENGTH=
Specifies the length of the entry made in the field specified with the FIELD operand (cf. list below). If an incorrect value is specified for LENGTH, no entry is made in the field specified with FIELD.
For system parameters of type C, information can be output to an output field which is too small, provided that only blanks are deleted from the contents of the system parameter (see also the system parameter DEFLUID, for example).

length
Length in bytes (see list below), specified as a decimal digit.

(r)
Register containing “length”.

MF=
For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix), see section “S-type macros” on page 29. The valid MF values are given at the start of the macro description under “Macro type” and are included in the macro format.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb).

List of information that can be requested

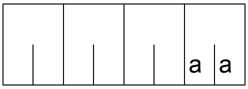
Identifier (info=)	**Length in bytes**	**Meanings of values output**
CONFNAME	8	Server type (model series) e.g.: S190-F (in the old format) If the server type is not entered in the system, 7.000␣␣␣ is output. *Note* In future, the server designations output in the old format may no longer be unique. Output in the new expanded format (INFO=CONFNAMX) is therefore recommended.
CONFNAMX	21	see **NSIINF** macro on page 702.
CPUID	64	Identifications of the CPUs. The output is split into 8 elements each 8 bytes in length: The n-th element contains the identification of the n-th CPU in hexadecimal form, as it is received from the hardware. If the n-th CPU is not available, the n-th element contains binary zero (X'00...00')
CPUSER	3	Serial number of the first CPU.
	6	Bytes 0- 2: serial number of first CPU. Bytes 3- 5: serial number of first CPU.
	12	Bytes 0- 2: serial number of first CPU. Bytes 3- 5: serial number of first CPU. Bytes 6- 8: serial number of third CPU. Bytes 9-11: serial number of fourth CPU. For any CPU not present, X'000000' is entered in the corresponding field. This entry has nothing to do with the CPU addresses.
HSIBASE	6	see **NSIINF** macro on page 702.
HSILINE	2	see **NSIINF** macro on page 702.
HSITYPE	4	The attributes of the current HSI type are output. Output: XS31 addressable memory area > 16 Mb; all addresses are interpreted as either 24-bit addresses or 31-bit addresses, depending on the addressing mode used.
HSIVM	2	see **NSIINF** macro on page 702.
MEMSIZE	4	Size of (physical) main memory available for software (in bytes).

Identifier (info=)	**Length in bytes**	**Meanings of values output**
OSAMODE	2	Provides information about the addressing mode of the operating system. Output: 31 31-bit addressing mode (address length = 31bits)
OSID	12	Bytes 0-7: program name of the operating system; e.g. 'BS2V190' . Bytes 8-11: version, e.g. 'V190' .
SYSBASE	4	Start address of the operating system in virtual address space.
<system parameter>		Value of the system parameter

Return information and error flags

Register R1 is overwritten.

R15:



A return code relating to the execution of the SINF macro is transferred in the rightmost byte of register R15.

X'aa'	**Meaning**
X'00'	The macro call was successful.
X'04'	Invalid address or invalid register. No action.
X'08'	Internal REQM call not executed. No action.
X'0C'	Invalid entry in INFO operand. No action.
X'10'	Invalid length entry for output field. No action.
X'14'	Illegal system parameter specified in INFO operand. No action.
X'18'	Incorrect macro format specified.

TABLE – Transfer load information

General

Application area:	Linking and loading; see page 47
Macro type:	24-bit interface: type R; see page 28 31-bit interface: type S (standard/E/L/C/D form); see page 29

The following applies when using the 31-bit interface:
No return code is transferred in the standard header.

- TABLE is replaced by the **ETABLE** and **ETABIT** macros.

Macro description

The user program employs the **TABLE** macro to transfer a table of entries to the dynamic binder loader DBL. These entries provide the DBL with information about the names and addresses of control sections (CSECT), entry points (ENTRY) and common areas (COMMON), enabling the DBL to use the transferred values to resolve external references, etc.

In particular, a user program which was linked by the TSOSLNK linkage editor, loaded with the ELDE static loader and which used the **LINK** macro can employ the **TABLE** macro to provide the DBL (called by means of the **LINK** macro) with information about the program structure. This prevents the DBL from reloading modules which have already been linked into the program by TSOSLNK.
During macro processing, a check is performed for each table entry as to whether the name of a load point is already known due to a previous load operation or a previous **TABLE** macro call. If the result of this check is positive, a warning is output and macro processing is continued normally.

Macro format 1 and description of operands

TABLE
$\left\{\begin{matrix}\text{addr}\\ \text{(r)}\end{matrix}\right\},\left\{\begin{matrix}\text{length}\\ \text{(r)}\end{matrix}\right\}$ [,PARMOD=24 / 31] [,MF=L / (E,..) / C / D]

Format 2

PBTABD
[,MF=(D,p) / (C,p)]

addr
Symbolic address of the table (see format below).

(r)
Register containing the address value “addr”. Register R1 must be used for the 24-bit interface.
If the 31-bit interface is used, the operand list address is loaded to register R1, which consequently is no longer available for storing user-specified information.

length
Specifies the length of the table in bytes.

(r)
Register containing “length”. Register R0 must be used for the 24-bit interface.

PARMOD=
Controls macro expansion. Either the 24-bit or the 31-bit interface is generated.
If PARMOD is not specified here, macro expansion is performed according to the specification for the **GPARMOD** macro or according to the default setting for the assembler (= 24-bit interface).

24
The 24-bit interface is generated. Data lists and instructions use 24-bit addresses (address space ≤ 16 Mb).

31
The 31-bit interface is generated. Data lists and instructions use 31-bit addresses (address space ≤ 2 Gb). Data lists start with the standard header.

MF=
For the 31-bit interface only:
The C-/D form is called with MF=C/D or MF=(C,p)/(D,p). p = prefix (up to 3 characters); default setting: p = I. The prefix only changes the field names (not the symbolic names for equates).If p = * is specified, the symbolic names are generated without a prefix. When the data area is assigned values dynamically, the initialization values for the standard header should be transfered from a data area generated with MF=L.

For a general description of the MF operand, its operand values and any subsequent operands (e.g. for a prefix) see section “S-type macros” on page 29. Valid MF values are given at the beginning of the macro description in “macro type” and in the call format.

The **PBTABD** macro generates a description of the table as a DSECT or data list for the 31-bit interface, where:

MF=
Specifies whether a DSECT or a data list is generated.
p = prefix (up to 3 chars.) of all symbolic names in the list.
If p = * is specified, names without prefix are generated.

D
A DSECT is generated; default setting.

C
A data list is generated.

Structure of the table for DBL

1. when the 24-bit interface is used:

Displacement	Length	Entry/function
0	1	X'02': identifies a CSECT or ENTRY X'03': identifies a labeled COMMON X'04': identifies a blank COMMON
1	8	name of the ENTRY, COMMON or CSECT (8 characters)
9	3	computed address
12	3	length of the COMMON (applies to COMMON only)

2. when the 31-bit interface is used:

Displacement	Length	Entry/function
0	1	X' F0' : identifies a CSECT X' F1' : identifies an ENTRY X' F3' : identifies a COMMON
1	8	name of the ENTRY, CSECT or COMMON (8 characters)

9	1	attributes: AMODE = 24 → X'02'
		AMODE = 31 → X'04'
		AMODE = ANY → X'06'

10	2	X'00' (required for alignment)
12	4	load address (31-bit-address)
16	4	length of the COMMON: otherwise 4 X'00'

The **PBTABD** macro generates the layout of this table.

Return information and error flags

A return code relating to the execution of the **TABLE** macro is transferred in register R15. Note that a different code is returned depending on the interface used (24-bit or 31-bit interface, as specified by means of the PARMOD operand or the **GPARMOD** macro).

if PARMOD=24:

R15:

						a	a

A return code relating to the execution of the TABLE macro is transferred in the rightmost byte of register R15.

Return code	Meaning
X'00'	Macro successfully executed.
X'04'	Macro not successfully executed; contents of register R1 deleted.

if PARMOD=31:

Register R1 is loaded with the address of the operand list.

R15:

c	c	b	b	a	a	a	a

A structured return code relating to the execution of the TABLE macro is transferred in register R15
(cc=Subcode2, bb=Subcode1, aaaa=Maincode)
where: cc=00: function executed normally
cc=0C: function not executed.

cc	bb	aaaa	Meaning
00	00	0000	Normal execution.
0C	00	0004	Parameter list not aligned.
0C	00	0008	Parameter list not assigned or address in register R1 outside address space.
0C	00	0014	A register number > 15 was specified.
0C	00	0018	Illegal utilization of a reserved field.
0C	00	0100	Error in length specification (“length” operand).
0C	00	0104	Incorrect name specification (table entry; displacement 1).
0C	00	0108	Wrong type specification; (table entry; displacement 0).
0C	00	010C	Incorrect AMODE specification (table entry).
0C	00	0110	Invalid load address specified (table load address). The address is outside class 6 memory.
0C	00	0200	System error.

cc	bb	aaaa	Meaning
0C	00	0300	Internal error during memory request ($REQM) or release of memory space ($RELM) (System error).
00	01	FFFF	Invalid specification for UNIT/FUNCTION in standard header.
00	03	FFFF	Invalid specification for VERSION in standard header.

6.2 Macros in alphabetical order

Macro	Function	SVC_{16}	Description
AINF	Measure resource utilization	63	page 168
ALESRV	Connect task with data space	0D	page 194
ALINF	Request information on access lists	0D	page 198
AMODE31	Transfer addressing mode	-	page 201
ARDS	Generate user accounting records	-	page 202
AREC	Write user accounting record	63	page 205
ASHARE	Load user' s shared code into memory pools	B7	page 210
ASPC	Enter memory allocation	63	page 222
AUDIT	Set audit mode	5F	page 224
BIND	Link and load load unit	B7	page 233
BKPT	Interrupt program run	5C	page 274
CALL	Load segments	-	page 276
CDUMP	Dump without program termination (31-bit interface) (24-bit interface)	 1A 19	page 1098
CDUMP2	Dump without program termination	1A	page 278
CHKEI	Check event item	7C	page 294
CHKPRV	Check privileges	31	page 297
CHKSI	Check serialization item	79	page 300
CLCOM	Terminate communication	36	page 304
CMD [1]	Call command (31-bit interface) (24-bit interface	 91 58	page 306
CONTXT	Access process data	80	page 324
CRYPT	Word encryption	10	page 340
CSTAT	Change page status (31-bit interface) (24-bit interface)	 01 4B	page 348
CSTMP	Set access type for memory pool (31-bit interface) (24-bit interface)	 01 7A	page 352
CTIME	Time stamp calculations	-	page 356
CUPAB	Address operand table	-	page 377
DCSTA	Display operand table for terminal attributes	-	page 381

Macro	Function	SVC$_{16}$	Description
DELFEI	Delete SOLSIG or POSSIG entry	BB	page 398
DEQAR	Dequeue access request	79	page 399
DISCO	Disable contingency definition	7B	page 404
DISEI	Disable event item	7C	page 407
DISMP	Disable memory pool (31-bit interface) (24-bit interface)	 01 7A	page 410
DISSI	Disable serialization item	79	page 414
DJINF	Create DSECTs or data areas for JIN macro	-	page 418
DJSI	Create DSECTs or data areas for job scheduler macros (24-bit interface)	-	page 421
DJSIPL	Create DSECTs or data areas for job scheduler macros (31-bit interface)	-	page 423
DPOFEI	Create POSSIG entry	BB	page 425
DSHARE	Unload shared code from memory pool	B7	page 431
DSOFEI	Create SOLSIG entry	BB	page 434
DSPSRV	Create or destroy data space	0D	page 439
DTMODE	Create DSECT or data list for TMODE macro	-	page 448
ENACO	Enable contingency definition	7B	page 451
ENAEI	Enable event item		page 454
ENAMP	Enable memory pool (31-bit interface) (24-bit interface)	 01 7A	page 458
ENASI	Enable serialization item	79	page 468
ENQAR	Enqueue access request	79	page 472
ENTER [1]	Initiate batch job (31-bit interface) (24-bit interface)	 91 58	page 477
ETABIT	Generate or change entry for symbol table	B7	page 496
ETABLE	Transfer load information	B7	page 499
EXIT	Terminate STXIT contingency process	80	page 507
GCCSN	Display CCS name for command and data input	27	page 510
GEPRT	Get program time (31-bit interface) (24-bit interface, type O) (type R)	 92 18 23	page 516

Macro	Function	SVC_{16}	Description
GETPRGV	Get program version	B7	page 520
GETSW	Get job switch	48	page 1107
GETUS	Get user switch	41	page 1108
GPARMOD	Control macro expansion	-	page 523
GTIME	Request date, time and zone information	-	page 525
HSITYPE	Output HSI attributes	87	page 1110
ILEMGT	ILE management	B7	page 537
ILEMIT	Generate or update list entry for ILE list	B7	page 543
IOSID	Specify operating system identification and version	AC	page 546
JINF	Request job information (31-bit interface) (24-bit interface)	 BF 51	page 549
JMGDJP	Create DSECT or data list for JMGJPAR macro	-	page 554
JMGJPAR	Output job parameters		page 555
JOBINFO	Request job information	8C	page 557
JSATTCH	Attach job scheduler to JMS (31-bit interface) (24-bit interface)	 BF 51	page 560
JSDETCH	Detach job scheduler from JMS (31-bit interface) (24-bit interface)	 BF 51	page 563
JSEXPCT	Request JSS events (31-bit interface) (24-bit interface)	 BF 51	page 565
JSINFO	Access STREAM-PARAMETER values (31-bit interface) (24-bit interface)	 BF 51	page 569
JSRUNJB	Transfer job to start (31-bit interface) (24-bit interface)	 BF 51	page 571
JSWAKE	Initiate timer event for job scheduler	BF	page 574
LDSLICE	Load slice	B7	page 576
LEVCO	Modify priority level of contingency process	7B	page 581
LGOFF [1]	Terminate job (31-bit interface) (24-bit interface)	 91 58	page 584
LKCAN	Cancel lock requests	C4	page 587

Macro	Function	SVC$_{16}$	Description
LKCVT	Convert existing lock	C4	page 590
LKDEQ	Dequeue lock	C4	page 598
LKENQ	Generate and enqueue lock	C4	page 602
LKEQU	Generate DLM-specific layouts	C4	page 612
LKINF	Output information on lock requests	C4	page 615
LKLSB	Generate layout of the Lock Status Block	C4	page 620
LPOV	Load segment	02	page 622
MINF	Output memory map	01	page 625
MRSINF [2]	Request MSCF informationen	7F	page 1112
MRSSTA [2]	Display MSCF status	85	page 1116
MSG7	Output message	60	page 1119
MSG7X	Output message	26	page 632
MSGRC	Output return codes for message macros	-	page 649
MSGSHOW	Output information about message files	60	page 652
MSGSINIT	Modify global area allocation list	60	page 656
MSGSMOD	Lock message files or add message files	60	page 658
NDGUINF	Output GS unit number	25	page 663
NKDINF	Output information on (peripheral) configuration	0E	page 667
NKGTYPE	Output device information	66	page 690
NSIINF	Output system information	87	page 702
NSIOPT	Output system parameters	87	page 710
OPCOM	Start ITC participation	32	page 717
OPSGEN	Control of S variable generation via MIP	C6	page 719
PASS	Wait one second	4C	page 722
PINF	Output global program information	B7	page 724
POSSIG	Post signal request	7C	page 736
RDATA	Read record from SYSDATA (31-bit interface) (24-bit interface)	 27 42	page 745
RDUID	Read user ID of current task	31	page 758
RELBF	Release receive queue	35	page 760
RELM	Release memory (31-bit interface) (24-bit interface)	 01 4A	page 761

Macro	Function	SVC$_{16}$	Description
RELMP	Release pages in memory pool (31-bit interface) (24-bit interface)	 01 7A	page 764
REQM	Request memory (31-bit interface) (24-bit interface)	 01 49	page 770
REQMP	Request pages in memory pool (31-bit interface) (24-bit interface)	 01 7A	page 774
RETCO	Return from contingency process	7B	page 780
RETRN	Return to program	-	page 781
REVNT	Receive event	34	page 783
RPOFEI	Send POSSIG signal	BB	page 789
RSOFEI	Request POSSIG signal	BB	page 791
SAVE	Save register	-	page 793
SEGLD	Load segments	-	page 797
SELPRGV	Select program version	B7	page 799
SETBF	Set buffer size	56	page 802
SETIC	Set interval timer	80	page 804
SETSW	Set job switch	47	page 1129
SETUS	Set user switch	40	page 1131
SEVNT	Send event	33	page 808
SINF	Output system information	87	page 1133
SOLSIG	Solicit signal request	7C	page 810
SRMUINF	Output entry from user catalog	B9	page 819
STAMCE	Read MRSCAT entries	21	page 833
STXIT	Set contingency interrupt address	80	page 869
SUSPEND	Suspend task	7B	page 881
SWITCH	Set and query job and user switches	2A	page 883
SYSFL [1]	Assign system files and TASKLIB (31-bit interface) (24-bit interface)	 91 58	page 894
SYSTA [1]	Output information on system file and TASKLIB assignment (31-bit interface) (24-bit interface)	 91 58	page 907

Macro	**Function**	**SVC$_{16}$**	**Description**
TABLE	Transfer load information (including PBTABD) (31-bit interface) (24-bit interface)	 B7 6F	page 1137
TCHNG	Modify terminal characteristics	46	page 910
TERM	Terminate program and procedure step	09	page 914
TINF	Output and modify task attributes	87	page 918
TMODE	Interrogate job attributes (31-bit interface) (24-bit interface)	 BF 46	page 926
TSPRIO	Output run priorities	-	page 932
TSTAT	Interrogate terminal attributes	46	page 933
TYPIO	Write message to console	96	page 943
UNBIND	Unload and unlink	B7	page 947
VMGINF	Output information on VM2000 operation	67	page 958
VPASS	Variable-length pass	59	page 963
VSVI1	Output link and load information	B7	page 965
VTCSET	Define logical control characters	-	page 991
VTSUCB	Create VTSU parameters for input and output	-	page 1023
WRCPT	Write checkpoint	05	page 1043
WRLST	Write record to SYSLST (31-bit interface) (24-bit interface)	 27 45	page 1049
WROUT	Write record to SYSOUT (31-bit interface) (24-bit interface)	 27 43	page 1053
WRTRD	Combined input/output at terminal (31-bit interface) (24-bit interface)	 27 44	page 1071

[1] MCLP macro

[2] Macro only for users of the HIPLEX MSCF software product

6.3 Macros arranged according to SVC number

SVC_{16}	SVC_{10}	Macros
01	01	CSTMP, DISMP, ENAMP, MINF, RELM, RELMP, REQM, REQMP, CSTAT (all 31-bit-interface)
02	02	LPOV
05	05	WRCPT
09	09	TERM
0D	13	ALESRV, ALINF, DSPSRV
0E	14	NKDINF
10	16	CRYPT
18	24	GEPRT (type O, 24-bit-interface)
1A	26	CDUMP2 (31-bit-interface)
21	33	STAMCE
23	35	GEPRT (type R)
25	37	NDGUINF
26	38	MSG7X
27	39	RDATA, WRLST, WROUT, WRTRD (all 31-bit-interface), GCCSN
2A	42	SWITCH
31	49	CHKPRV, RDUID
32	50	OPCOM
33	51	SEVNT
34	52	REVNT
35	53	RELBF
36	54	CLCOM
40	64	SETUS
41	65	GETUS
42	66	RDATA (24-bit-interface)
43	67	WROUT (24-bit-interface)
44	68	WRTRD (24-bit-interface)
45	69	WRLST (24-bit-interface)
46	70	TMODE (24-bit-interface), TCHNG, TSTAT
47	71	SETSW
48	72	GETSW
49	73	REQM (24-bit-interface)

SVC_{16}	SVC_{10}	Macros
4A	74	RELM (24-bit-interface)
4C	75	CSTAT (24-bit-interface)
4C	76	PASS
51	81	JINF, JSATTCH, JSDETCH, JSEXPCT, JSINFO, JSRUNJB (all 24-bit-interface)
56	86	SETBF
58	88	CMD, ENTER, LGOFF, SYSFL, SYSTA (all 24-bit-interface)
59	89	VPASS
5C	92	BKPT
5F	95	AUDIT
60	96	MSG7, MSGSINIT, MSGSMOD, MSGSHOW
63	99	AINF, AREC, ASPC
66	102	NKGTYPE
67	103	VMGINF
6F	111	TABLE (24-bit-interface)
79	121	ENASI, DISSI, ENQAR, DEQAR, CHKSI
7A	122	ENAMP, DISMP, REQMP, RELMP, CSTMP (all 24-bit-interface)
7B	123	DISCO, ENACO, LEVCO, RETCO, SUSPEND
7C	124	CHKEI, DISEI, ENAEI, POSSIG, SOLSIG
80	128	CONTXT, EXIT, STXIT, SETIC
87	135	HSITYPE, NSIINF, NSIOPT, SINF, TINF
8C	140	JOBINFO
91	145	CMD, ENTER, LGOFF, SYSFL, SYSTA (all 31-bit-interface)
92	146	GEPRT (31-bit-interface)
96	150	TYPIO
AC	172	IOSID
B7	183	TABLE (31-bit-interface), ASHARE, BIND, DSHARE, ETABIT, ETABLE, GETPRGV, ILEMGT, ILEMIT, LDSLICE, PINF, SELPRGV, UNBIND, VSVI1
B9	185	SRMUINF
BB	187	DELFEI, DPOFEI, DSOFEI, RPOFEI, RSOFEI
BF	191	JINF, JSATTCH, JSDETCH, JSEXPCT, JSINFO, JSRUNJB, JSWAKE, TMODE (all 31-bit-interface)
C4	196	LKCAN, LKCVT, LKDEQ, LKENQ, LKEQU, LKINF, LKLSB
C6	198	OPSGEN

6.4 Other macros in BS2000 OSD/BC

This table shows all the macros in BS2000 OSD/BC that are not described in the present manual. In addition to a brief description of the macro's function, the table also indicates the manual in which the macro is treated in detail.

Macro	Brief description	Manual
ADDPLNK	Define pool link name	[7]
BINDER	Call the BINDER as subprogram	[5]
BTAM	Complete user requirements for BTAM	[7]
CALENDR	Create, modify and output calendar data	[6]
CATAL	Process catalog entry	[7]
CHKFAR	Check access rights to a file	[7]
CHNGE	Change TFT entry	[7]
CLOSE	Close file	[7]
COMPFIL	Compare two files	[7]
COPFILE	Copy file	[7]
CREAIX	Create secondary key for ISAM file	[7]
CREPOOL	Create ISAM pool	[7]
DECFILE	convert encrypted file into unencrypted file	[7]
DELAIX	Delete secondary key for ISAM file	[7]
DELPOOL	Delete or release ISAM pool	[7]
DIV	Process file via “window” in virtual address space	[7]
DROPTFT	release TFT entry	[7]
EAM	Complete user requirements for EAM	[7]
ELIM	Erase record from ISAM file	[7]
ENCFILE	convert unencrypted file into encrypted file	[7]
ERASE	Erase file(s)	[7]
EXCALL	Call exit routines from TU programs	[31]
EXLST	Apply exit address list for error routines	[7]
EXRTN	Return from error routines	[7]
FCB	Apply file control block	[7]
FCBAD	Apply FCB addresses	[7]
FCEKIO	Test status of an asynchronous input/output; execute final test if the input/output is terminated	[32]

Macro	**Brief description**	**Manual**
FCEKSG	Test status of a requested device signal and execute final processing if the signal is received	[32]
FCLSDV	Terminate device operation	[32]
FDECCP	Create and store channel program	[32]
FDEFIO	Define device tasks	[32]
FDELCP	Delete channel program	[32]
FDEVIB	Create device information block	[32]
FEOV	Close volume and introduce volume change	[7]
FEXCIO	Start input/output and do not wait for termination (equivalent to asynchronous input/output)	[32]
FEXIOW	Start input/output and wait for termination (equivalent to synchronous input/output)	[32]
FHLTIO	Force termination of asynchronous input/output	[32]
FILE	Define file characteristics and control file processing	[7]
FILELST	Create variable operand areas for FILE macro	[7]
FOPNDV	Open device operation	[32]
FPAMACC	Formulate FASTPAM file access	[7]
FPAMSRV	Formulate FASTPAM administration calls	[7]
FRECSG	Request delivery of the next device signal	[32]
FSTAT	Request catalog information	[7]
FTIMOT	Define max. time to wait for the end of the input/output	[32]
FWFSG	Wait for the chronologically next device signal and execute final processing	[32]
FWFTIO	Wait for end of an asynchronous input/output and execute final processing	[32]
GET	Read record from file	[7]
GETFL	Search for marking in file and read record	[7]
GETINSP	Output installation path	[33]
GETINSV	Output version of the installation unit	[33]
GETKY	Read record with specified key	[7]
GETPROV	Output selected product version	[33]
GETR	Read next record in direction of start of file	[7]
IDBPL	Generate DSECT for BTAM action macro	[7]
IDFCB	Generate DSECT for FCB	[7]
IDFCBE	Generate DSECT for FCB extension	[7]

Macro	Brief description	Manual
IDMCB	provide MFCB (EAM control block with symbolic name)	[7]
IDPPL	UPAM: PAM operand list	[7]
IMOS...	Macros of IMON-BAS	[33]
IMOK...	Macros of IMON-SIC	[33]
IMPNFIL	Create a catalog entry for node files (import)	[7]
IMPORT	Import private files or data media	[7]
INSRT	Insert record in file	[7]
ISREQ	Release record, area or data block bar	[7]
LBRET	Return from user label routines	[7]
LFFSNAP	Dateien von einem Snapset auflisten	[7]
LJFSNAP	Jobvariablen von einem Snapset auflisten	[7]
MAILFIL	Datei per E-Mail an eine Benutzerkennung versenden	[7]
MCSINFO	Fetch information on HIPLEX-MSCF configuration	[26]
NBMAP	Describe message scope	[10]
NBMHE	Describe format of message header	[10]
NDWERINF	Query status byte	[7]
OPEN	Open file	[7]
OSTAT	Output information on opened files	[7]
PAM	Execute UPAM actions	[7]
PRNT...	Macros for outputting files	[23]
PUT	Write record to file	[7]
PUTX	Replace record in file	[7]
RDTFT	Request information from TFT and TST	[7]
RELTFT	Delete TFT entry	[7]
RELSE	Close data block and release buffer	[7]
REMPLNK	Delete pool link name	[7]
RETRY	Repeat macro	[7]
RFFSNAP	Dateien von einem Snapset restaurieren	[7]
RJFSNAP	Jobvariablen von einem Snapset restaurieren	[7]
SELPROV	Select product version	[33]
SETINSP	Enter or modify installation path	[33]
SETL	Position internal file record pointer	[7]
SHOPLNK	Output information on ISAM pool link name	[7]

Macro	**Brief description**	**Manual**
SHOPOOL	Output information on ISAM pool	[7]
SHOWAIX	Output information on secondary key	[7]
SPSINF	Output information on SPOOL parameter file	[23]
STORE	Transfer record to position in file defined by key	[7]
VERIF	Recreate file	[7]

6.5 Standardized function key codes

The first column (CODE) is divided into
a) the standard function key code
b) the function key code if the screen is erased before input.

Additional notes
1) Reserved for ESCAPE/BREAK function if TIAM access method is used.
2) For default values see VTSU operating parameter COMPKEYS in “VTSU” manual [30].

Code		**Meaning**		**8110**	**8151**	**8152**	**816x**	**974x** **9750** **9752** **9755**	**9756** **9758** **9759** **9762** **9763**	**3270 2)**
a)	**b)**									
00	10	Data communication		DÜ	DÜZ, DÜM	DÜZ,DÜM ,	DÜ, DÜ1,	DÜ, DÜ1,	SEND	ENTER
01		(normal)	1		FT1	DÜB	DÜ2	DÜ2	K1	PA1
02			2		FT2	F1	K1	K1	K2 1)	PA2 1)
03			3		1)	F2 1)	K2 1)	K2 1)	K3	PA3,PF6
04			4		DVA	F3	K3	K3	K4	PF7
05			5			F1+FZ	K4	K4	K5	PF8
06			6			F2+FZ 1)	K5	K5	K6	PF9
07			7			F3+FZ	K6	K6	K7	PF10
08		Short message	8				K7	K7	K8	PF11
09			9				K8	K8	K9	PF12
0A			10				K9	K9	K10	PF13
0B			11				K10	K10	K11	PF14
0C			12				K11	K11	K12	PF15
0D			13				K12	K12	K13	PF16
0E			14				K13	K13	K14	PF17
							K14	K14		
10		Erase memory								CLEAR

Code		Meaning		8110	8151	8152	816x	974x 9750 9752 9755	9756 9758 9759 9762 9763	3270 2)
a)	b)									
20	30	Data communication (marked)	0			DÜZ, DÜM, DÜB+FZ		DÜZ, DÜM, DÜB+FZ		
21	31		1				F1		F1	PF1
22	32		2				F2		F2	PF2
23	33		3				F3	F1	F3	PF3
24	34		4				F4	F2	F4	PF4
25	35		5				F5	F3	F5	PF5
26	36		6					F4	F6	PF18
27	37		7					F5	F7	PF19
28	38		8						F8	PF20
29	39		9						F9	PF21
2A	3A		10						F10	PF22
2B	3B		11						F11	PF23
2C	3C		12						F12	PF24
2D	3D		13						F13	
2E	3E		14						F14	
2F	3F		15						F15	
30		Data communication (position data)				PU				
40	50	Data communication (special)	0				Bypass input			
41	51		1				Pos. response			
42	52		2				Neg. response			
43	53		3							
45	55	Magnetiic ID Reader						X	X	X
46	56	Data communication (marked)	16						F16	
47	57		17						F17	
48	58		18						F18	
49	59		19						F19	
4A	5A		20						F20	
4B	5B		21						F21	
4C	5C		22						F22	
4D	5D		23						F23	
4E	5E		24						F24	
60		Chip card date							X	
80		ID stuck						X	X	X

Abbreviations

AFZ	Delete lines (Ausfügen Zeile) - terminal keyboard
AID	Advanced Interactive Debugger
AL	Access List (for data spaces)
ALE	Access List Entry
ALET	Access List Entry Token (pointer to data space)
AR	Access Register (for data spaces)
ASCII	American Standard Code of Information Interchange
BTAM	Basic Tape Access Method
CCS	Coded character set (code table)
CJC	Conditional Job Control
CNS	Communication Network System
CPU	Central Processing Unit
CLT	Communication Link Table
CLTF	Common Log Task Facility (error source for macros)
CSECT	Control SECTion
DBL	Dynamic Binder Loader
DCAM	Data Communication Access Method
DCM	Data Communication Methods
DIV	Data In Virtual (access method for data spaces)
DLAM	Dynamic Loadable Access Method
DLM	Distributed Lock Manager
DMS	Data Management System
DRV	Dual Recording by Volume
DSECT	Dummy Section
DSSM	Dynamic SubSystem Management (Subsystemverwaltung)
EAM	Evanescent Access Method
EBCDIC	Extended Binary Coded Decimal Interchange Code

ECR	Executive Communication Region
EFZ	Insert lines (Einfgen Zeile) - terminal keyboard
ELDE	Static loader in BS2000
EOF	End Of File
ES	Event switch
ESA	Enterprise System Architecture (extended addressing with data spaces)
ESD	External Symbol Dictionary
ETX	End of TeXt
FCB	File Control Block
FCP	File Control Processor
FGG	File Generation Group
FIFO	First In First Out
FM	File Maintenance
FMS	File Management System
FT	File Transfer
Gb	Gigabyte; 1 Gb = 1024 Mb = 1.048.576 Kb = 1.073.741.824 bytes
GR	General Register
GS	Global Storage
HDR1	HeaDeR label 1
HIPLEX MSCF	HIPLEX Multi System Control Facility (software product)
IDA	Interactive Debugging Aid (old debugging aid)
II	Information Indicator
IOC	Input/Output Controller
IOP	Input/Output Processor
ISAM	Indexed Sequential Access Method (file structure/access method)
ISD	Internal Symbol Dictionary
ITC	InterTask Communication
ITN	Internal Task Number
JCB	Job Control Block
JSS	Job Scheduling Supports
JTBP	Job To Be Processed Block
JTBX	Job To Be Processed Extension
JV	Job Variable

K	Kilobyte; 1K = 1024 bytes
LIFO	Last In First Out
LMR	Library Maintenance Routine
LMS	Library Maintenance System
LPS	Delete screen (Löschen Bildschirmspeicher) - terminal keyboard
LZE	Logical line end (Logisches Zeilenende) terminal keyboard
Mb	Megabyte; 1 Mb = 1,024 Kb = 1,048,576 bytes
MC	MRS communication
MCLP	Macro Command Language Processor
MOPS	Million Operations Per Second
MP	Memory Pool
MRS	Multiprocessor system (Mehrrechnersystem)
MRSCAT	MRS CATalog
MSCF	Multi System Control Facility (software product)
MPVS	Multiple Public Volume Set
NDM	Nucleus Device Management
NTL	No Time Limit
OPR	Overwrite PRotection
PAM	Primary Access Method (file structure/access method)
PC	Program Counter
PCB	Process Control Block
PCR	Program Control Register
PL/1	Programming Language 1
PLI1	PL/1 Compiler
PT	Program Table
PVS	Public Volume Set
RC	Return Code (return information for macro processing)
RFA	Remote File Access
RS	Return Switch
RTIO	Remote Terminal Input Output
SAM	Sequential Access Method (file structure/access method)
SI	Secondary Indicator
SPD	Shareable Private Disk

SPID	SPace IDentification (for data spaces)
SPL	Software Programming Language
SPOOL	Simultaneous Peripheral Operation OnLine
SVC	SuperVisor Call (Assembler command, privileged)
TCB	Task Control Block
TCS	TeleCommunication System
TFT	Task File Table
TIAM	Terminal Interactive Access Method
TOD	Time Of Day
TODR	Time Of Day clock Register
TODX	Extended Time Of Day clock Register
TOS	Tape Operating System
TSET	Tape SET
TSN	Task Sequence Number
TSOS	Time Sharing Operation System
TPR	Task Privileged
TU	Task Unprivileged
UCON	Universal Console
UPAM	User-PAM
USASCII	USA Standard Code of Information Interchange
UTM	Universal Transaction Monitor
VFB	Vertical Format Buffer
VM	Virtual Memory
VSN	Volume Serial Number
VTOC	Volume Table of Contents
VTSU-B	Virtual Terminal Support - Basic
XCS	Cross Coupled System

Related publications

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

[1] **Assembler Instructions** (BS2000)
Reference Manual

[2] **ASSEMBH** (BS2000)
Reference Manual

[3] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of ASSEMBH Programs
User Guide

[4] **BLSSERV** (BS2000)
Dynamic Binder Loader / Starter
User Guide

[5] **BINDER** (BS2000)
Binder in BS2000
User Guide

[6] **BS2000 OSD/BC**
CALENDAR
User Guide

[7] **BS2000 OSD/BC**
DMS Macros
User Guide

[8] **BS2000 OSD/BC**
Introductory Guide to DMS
User Guide

[9] **BS2000 OSD/BC**
Diagnostics Handbook
User Guide

[10] **BS2000 OSD/BC**
Introduction to System Administration
User Guide

[11] **BS2000 OSD/BC**
System Installation
User Guide

[12] **DSSM/SSCM**
Subsystem Management in BS2000
User Guide

[13] **BS2000 OSD/BC**
Accounting Records
User Guide

[14] **SECOS** (BS2000)
Security Control System
User Guide

[15] **DCAM** (BS2000)
Macros
User Guide

[16] **TIAM** (BS2000)
User Guide

[17] **VM2000** (BS2000)
Virtual Machine System
User Guide

[18] **BS2000 OSD/BC**
System Managed Storage
User Guide

[19] **BS2000 OSD/BC**
Commands
User Guides

[20] **SDF-A** (BS2000)
System Dialog Facility - Administration
User Guide

[21] **SDF-P** (BS2000)
Programming in the Command Language
User Guide

[22] **JV** (BS2000)
Job Variables
User Guide

[23] **BS2000 OSD/BC**
Spool & Print - Macros and Exits
User Guide

[24] **RSO** (BS2000)
Remote SPOOL Output
User Guide

[25] **DRV** (BS2000)
Dual Recording by Volume
User Guide

[26] **HIPLEX MSCF** (BS2000)
BS2000 Processor Networks
User Guide

[27] **BS2000 OSD/BC**
Utility Routines
User Guide

[28] **MAREN** (BS2000)
Volume 1: Basics of MTC Management
Volume 2: User Interfaces
User Guide

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

[30] **VTSU**
Virtual Terminal Support
User Guide

[31] **BS2000 OSD/BC**
System Exits
User Guide

[32] **ADAM** (BS2000)
Abstract Device Access Method
User Guide

[33] **IMON** (BS2000)
Installation Monitor
User Guide

[34] **BS2000**
User Commands (ISP Format)
User Guide

For compatibility reasons only.

Index