z/OS
2.5

DFSMS Macro Instructions for Data Sets

SC23-6852-50

Note

Before using this information and the product it supports, read the information in “Notices” on page 415.

This edition applies to Version 2 Release 5 of z/OS® (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2023-06-26

Contents

Figures

Tables

About this book

This book is intended to help you use virtual storage access method (VSAM) and non-VSAM IBM® data management macros to process data sets. “Part 1. VSAM Macro Instructions” describes virtual storage access method (VSAM) macros, examples of coding the macros in assembler language, and the return codes. “Part 2. Non-VSAM Macro Instructions” describes non-VSAM macros and the return codes. Each part presents the macros in alphabetical order. The standard form of each macro is described first, followed by the list and execute forms, if available. The list and execute forms are available only for macros that pass parameters in a list.

Use this book with *z/OS DFSMS Using Data Sets* which describes the access methods and how to write programs that process VSAM and non-VSAM data sets.

Macros allow you to communicate service requests to the access method routines. The macros are placed in the macro library when the operating system is installed. The assembler expands each macro into executable machine language instructions or data, and shows the exact macro expansion in the assembler listing. The executable instructions typically consist of branches around data fields, load register instructions, and either branch instructions or supervisor calls (SVC) that transfer control to the proper program. The data fields in each macro are parameters that are passed to the access method routine.

The operation of most macros depends on the options you select when coding the macro. For these macros, separate descriptions are provided for each parameter, keyword, and option. The standard, list, and execute forms of the macros are provided where differences exist; otherwise, just the standard form is provided.

The macros described in this book are in the standard system macro library, SYS1.MACLIB. To write programs that use z/OS MVS supervisor services refer to the following books:

- *z/OS MVS Programming: Authorized Assembler Services Guide*
- *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*
- *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*
- *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*
- *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO*
- *z/OS DFSMSdfp Advanced Services*
- DFSMS macros require High Level Assembler.
- See *z/OS DFSMS Introduction* for DFSMS requirements.

To learn about catalogs and the access method services commands, see:

- *z/OS DFSMS Access Method Services Commands*, which describes the access method services commands used to process VSAM data sets.
- *z/OS DFSMS Managing Catalogs*, which describes how to create master and user catalogs.

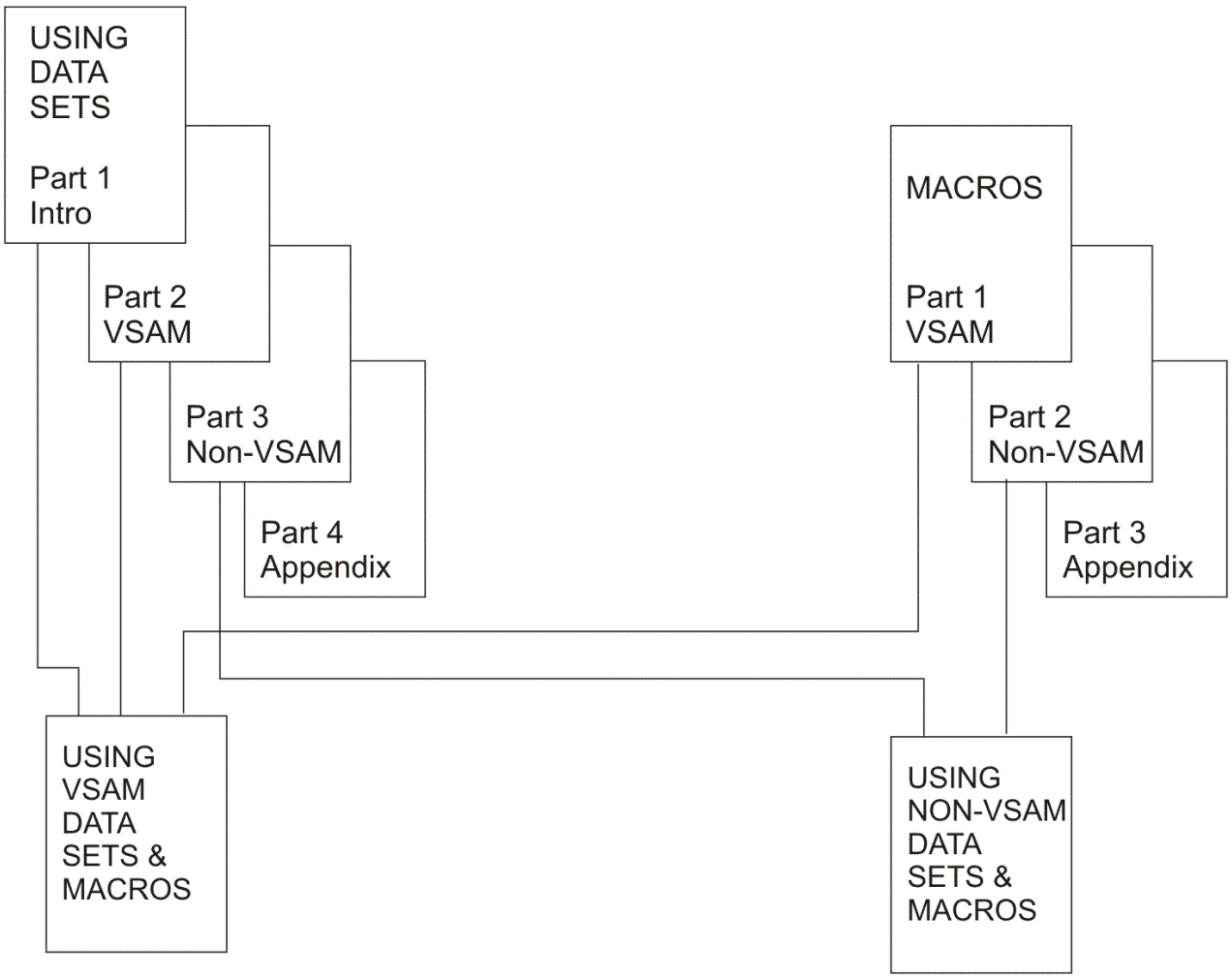
Use of the following access methods is not recommended and alternatives are suggested.

- BISAM (use VSAM instead)
- QISAM (use VSAM instead).

The Mass Storage System (MSS) and the ACQRANGE, CNVTAD, MNTACQ macros are no longer supported.

Preparing your books for use

All the VSAM and non-VSAM guidance material is in *z/OS DFSMS Using Data Sets*. All the macros are in *z/OS DFSMS Macro Instructions for Data Sets*. However, you can rearrange the sections of these books to create your own VSAM guide and reference and non-VSAM guide and reference.



Use the following steps to create the VSAM book:

1. Remove all of Part 1 and Part 2 from *z/OS DFSMS Using Data Sets*. Make a copy of Part 1 for the non-VSAM book.
2. Remove all of Part 1 from *z/OS DFSMS Macro Instructions for Data Sets*.
3. Select the appendixes you want for the VSAM book.
4. Reassemble all pages in a three- or five-ring binder.

Use the following steps to create the non-VSAM book:

1. Remove all of Part 3 from *z/OS DFSMS Using Data Sets* and all of Part 2 of *z/OS DFSMS Macro Instructions for Data Sets*.
2. Select the appendixes you want for the non-VSAM book.
3. Reassemble all pages in a three- or five-ring binder.

Required product knowledge

To use this book effectively, you should be familiar with the following:

- Assembler language
- Catalog administration
- Job control language
- VSAM and non-VSAM data management

Notational conventions

A uniform notation describes the format of data management macro instructions. This notation is not part of the language; it is merely a way of describing the format of the instructions. The instruction format definitions in this book use the following conventions:

[ ]
Brackets enclose an optional entry. You may, but need not, include the entry. Examples are:

- [*length*]
- [MF=E]

|
An OR sign (a vertical bar) separates alternative entries. You must specify one, and only one, of the entries unless you allow an indicated default. Examples are:

- [REREAD|LEAVE]
- [*length*|'S']

{ }
Braces enclose alternative entries. You must use one, and only one, of the entries. Examples are:

- BFTEK={S|A}
- {K|D}
- {*address*|S|0}

Sometimes alternative entries are shown in a vertical stack of braces. An example is:

```
MACRF={{(R[C|P])}
{(W[C|P|L])}
{(R[C],W[C])}}
```

In the example above, you must choose only one entry from the vertical stack.

...
An ellipsis indicates that the entry immediately preceding the ellipsis may be repeated. For example:

- (*dcbaddr*,[(*options*)],. . .)

‘ ’
A **‘ ’** indicates that a blank (an empty space) must be present before the next parameter.

UPPERCASE BOLDFACE
Uppercase boldface type indicates entries that you must code exactly as shown. These entries consist of keywords and the following punctuation symbols: commas, parentheses, and equal signs. Examples are:

- CLOSE , , , ,TYPE=T
- MACRF=(PL,PTC)

UNDERSCORED UPPERCASE BOLDFACE
Underscored uppercase boldface type indicates the default used if you do not specify any of the alternatives. Examples are:

- [EROPT={ACC|SKP|ABE}]
- [BFALN={F|D}]

Lowercase Italic
Lowercase italic type indicates a value to be supplied by you, the user, usually according to specifications and limits described for each parameter. Examples are:

- *number*
- *image-id*
- *count*

Macro format

Data management macros follow the rules of assembler language and are written in the following format:

Name	Operation	Operands (Parameters)	Comments
Symbol or blank	Macro name	None, one or more operands separated by commas	

Use the operands to specify services and options you need and code them according to the following general rules:

- If the operand is a combination of bold capital letters and italic lowercase letters (for example, LRECL=*absexp*), code the capital letters and equal sign exactly as shown and substitute the appropriate address, name, or value for the italic lowercase letters.
- Code commas and parentheses exactly as shown.

  Omit the comma that follows the last operand in a statement. Brackets and braces show how to use commas and parentheses the same way they show how to use operands.
- Several macros contain the name `'S'`. Use the apostrophe on both sides of the S operand.

If you need to substitute a name, value, or address, the notation you use depends on the operand you are coding. The following two examples show how an operand can be coded:

DDNAME=*symbol*
In this example, you can only code a valid assembler-language symbol for the operand.

***dcb address*-RX-Type Address, (2-12), or (1)**
In the above example, you can substitute an RX-type address, any general register 2 through 12, or general register 1.

The following examples show what each notation means and how you can code an operand:

symbol
Any valid assembler-language symbol, which is an alphabetic character followed by 0–61 alphanumeric or national characters, with no special characters except underscore and no blanks.

decimal digits
Any decimal digits up to the maximum value allowed for the specific operand. If both symbol and decimal digit are used, an absolute expression is also allowed.

(2-12)
Any of the general registers 2 through 12, coded in parentheses, to distinguish the register number from an A-type address. For example, if you code register 3, use the form `(3)`. The following is an example with the CLOSE macro:

```
CLOSE   ((3))
```

If you want to use one of the registers 2 through 12, code it as a decimal number, a symbol (equated to a decimal number), or an expression that yields a value of 2 through 12.

(1)
You can use general register 1 as an operand. Specify the register as (1). When register 1 is used as an operand, the instruction that loads the parameter value into the register is not included in the macro expansion.

(0)
You can use general register 0 as an operand. Specify the register as (0). When register 0 is used as an operand, the instruction that loads the parameter value into the register is not included in the macro expansion.

RX-Type Address
Any valid assembler-language RX-type address. The following shows examples of each valid RX-type address:

Name	Operation	Operand
ALPHA1	L	1,39(4,10)
ALPHA2	L	REG1,39(4,TEN)
BETA1	L	2,ZETA(4)
BETA2	L	REG2,ZETA(REG4)
GAMMA1	L	2,ZETA
GAMMA2	L	REG2,ZETA
GAMMA3	L	2,=F'1000'
LAMBDA1	L	3,20(,5)

Both ALPHA instructions specify explicit addresses; REG1 and TEN are absolute symbols. Both use index registers. Both BETA instructions specify implied addresses. Indexing is omitted from the BETA and GAMMA instructions. GAMMA1 and GAMMA2 specify implied addresses. The second operand of GAMMA3 is a literal. LAMBDA1 specifies an explicit address with no indexing.

A-Type Address
Any address that can be written as a valid assembler-language A-type address constant. You can write an A-type address constant as an absolute value, a relocatable symbol, or a relocatable expression. Operands that require an A-type address are inserted into an A-type address constant during the macro expansion process.

absexp
An absolute value or expression. An absolute expression can be an absolute term or an arithmetic combination of absolute terms. An absolute term can be a nonrelocatable symbol, a self-defining term, or the length attribute reference.

relexp
A relocatable symbol or expression. A relocatable symbol or expression is one whose value changes by n if the program in which it appears is relocated n bytes away from its originally assigned area of storage.

Rules for register usage

Many macro expansions include instructions that assume a base register previously defined by a USING statement. The USING statement must establish addressability so that the macro expansion can include a branch around the in-line parameter list, if present, and list the data fields and addresses specified in the macro operands.

Macros that use a BAL, BALR, BAS, or BASR instruction to pass control to an access method routine normally require that register 13 contain the address of an 18-word register-save area. The READ, WRITE, CHECK, GET, and PUT macros are of this type. If a macro requires a save area and your program calls the macro in 31-bit mode, the register 13 contents must be a valid 31-bit address and it may point above the 16 MB line.

Macros that use a supervisor call (SVC) instruction to pass control to an access method routine might modify general registers 0, 1, 14, and 15 without restoring them. Unless otherwise specified in the macro description, the contents of these registers are undefined when the system returns control to the problem program.

When an operand is specified as a register, the problem program must have inserted the value or address to be used into the register as follows:

- Unless the macro description states otherwise, and the register is to contain a value, that value must be placed in the low-order portion of the register. Any unused bits in the register should be set to zero.
- If the register is to contain a 24-bit address, the address must be placed in the low-order 3 bytes of the register, and the high-order byte of the register should be set to zero.
- If the register is to contain a 31-bit address, the address must be placed in the low-order 31 bits of the register, and the high-order bit of the register should be set to zero.

Note that, if the macro accepts the RX-type address, an efficient way to clear the high-order part of a register is to code the parameter as 0(,reg) rather than merely as (reg). Then, the macro expands as:

`LA parmreg,0(,reg)` by macro rather than:

`LA reg,0(,reg)` by user and `LR parmreg,reg` by macro.

If your program is executing in 24-bit addressing mode, this clears the high-order byte. In 31-bit mode this clears the high order. You will get incorrect results if the number of the source register is zero.

Environmental considerations

To generate code that is correct for the environment in which the program runs, some macros need to know one or more of the following characteristics of that environment:

- The addressing mode (AMODE) at the time the macro is issued
- The ASC mode of the program at the time the macro is issued
- The architectural level in which the program runs.

In addition some macros generate more efficient code when the execution environment is newer. For macros that are sensitive to their environment, use the SYSSTATE macro to define the environment. During the assembly stage, SYSSTATE sets global symbols. Later in your source code the macro checks the global symbols and generates the correct code, which might mean avoiding using a z/Architecture instruction or an access register. For more information about SYSSTATE, refer to *z/OS MVS Programming: Assembler Services Reference IAR-XCT*.

IBM recommends that you issue the SYSSTATE macro before you issue other macros. Once a program has issued SYSSTATE, there is no need to reissue it, unless the program switches from one AMODE to another or one ASC mode to another or has code paths that are isolated according to architecture level or operating system release. If you switch AMODE or ASC mode to a different architecture code path, issue SYSSTATE immediately after the switch to indicate the new state. In general, specify SYSSTATE ARCHLVL=1 and switch to SYSSTATE ARCHLVL=2 before issuing macros in sections of code that run in z/Architecture mode. If you do not issue the SYSSTATE macro, the system assumes the macro is issued:

- In AMODE other than 64-bit. No macro documented in this information can be issued in 64-bit mode.
- In primary ASC mode
- In ESA/390 architectural level.

All VSAM macros can be issued in 24-bit or 31-bit mode. The non-VSAM macro descriptions state whether it can be issued in 31-bit mode and which fields may reside above the 16 MB line. For information about which macros can be issued in 31-bit mode, refer to Appendix A, “Macros available by access method,” on page 371.

For those macros that may be issued in 31-bit addressing mode, the macro description may state that when it is issued in 31-bit addressing mode, it expects all addresses to be valid 31-bit addresses. A valid, or clean, 31-bit address is a 4-byte address in which, when referring to location below the 16 MB line, the high order byte is zero, or, when referring to locations above the 16 MB line, the high order bit is zero. For more information, refer to “Data above the 16MB line” on page 150.

Rules for continuation lines

You can continue the operand field of a macro on one or more additional lines as follows:

1. Enter a continuation character (not blank, and not part of the operand coding) in column 72 of the line.
2. Continue the operand field on the next line, starting in column 16. All columns to the left of column 16 must be blank. Comments may be continued after column 16.

Note that if column 72 is filled in on one line and you try to continue an operand or start a new statement after column 16 on the next line, this statement will be taken as a comment belonging to the previous statement.

You can code the operand field that is being continued in one of two ways: 1) Code the operand field through column 71, with no blanks, and continue in column 16 of the next line; or 2) truncate the operand field by a comma, where a comma normally falls, with at least one blank before column 71, and then continue in column 16 of the next line. Figure 1 on page xxiii shows an example of each method.

```
-------------------------------------------------------------------------------------------
Name          Operation     Operands        Comments
1             10            16              41                                  72
-------------------------------------------------------------------------------------------
NAME1         OP1           OPERAND1,OPERAND2,OPERAND3,OPERAND4,OPERAND5,OPERAND6,OPEX
                            RAND7             THIS IS METHOD 1




NAME2         OP2           OPERAND1,         THIS IS METHOD 2                  X
                            OPERAND2,
                            OPERAND3,
                            OPERAND4,
                            OPERAND5,
                            OPERAND6
NAME3         OP3           OPERAND1,         THIS IS ANOTHER EXAMPLE           X
                            OPERAND2,         OF METHOD 2
                            OPERAND3
```

Figure 1. Continuing the operand field

z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

To find the complete z/OS library, go to IBM Documentation (www.ibm.com/docs/en/zos).

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xxvii.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM Documentation function
If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdocs@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS DFSMS Macro Instructions for Data Sets, SC23-6852-50
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Note: IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under IBM z/OS Product Documentation Update Policy (www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument).

Summary of changes for z/OS Version 2 Release 5 (V2R5)

New

The following content is new.

August 2022 refresh

- For APAR OA59251, new reason code 215(X'D7') is added. For more information, see Table 17 on page 124. (APAR OA59251, which also applies to z/OS V2R4 and V2R3)

April 2022 refresh

- For APAR OA61850, a new return code, 32, is added to register 15. For more information, see "STOW completion codes" on page 339.
- For APAR OA62361, including z/OS V2R4, "DCB—Construct a data control block (QSAM)" on page 212 is updated.

February 2022 refresh

- For APAR OA60658, "DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)" on page 230 is updated with BYPASS_EXTENT_CHECK.

January 2022 refresh

- For APAR OA60182, reason code 187(X'BB') is updated. For more information, see "Reason code (logical errors)" on page 124.

Prior to January 2022 refresh

- New reason code 163(X'A3') is added in the ACBERFLG field of the ACB in support of DFSMS archived key support. See Table 8 on page 108.

Changed

The following content is changed.

June 2023 refresh

- Reason code 82(X'52') is added to Table 8 on page 108. (APAR OA58995, which also applies to z/OS V2R4)

May 2023 refresh

- ZHYPERWRITE keyword is added to:
  - "ACB—Generate an access method control block at assembly time" on page 9
  - "GENCB—Generate an access method control block at execution time" on page 32
  - "GENCB—Generate a request parameter list at execution time" on page 40
  - "RPL—Generate a request parameter list at assembly time" on page 70

- – “MODCB—Modify an access method control block” on page 54
- – “MODCB—Modify a request parameter list” on page 57

(APARs OA62148 and OA62149, which also apply to z/OS V2R4.)

- Keyword LL for BLDL is updated. For more information, see “BLDL—Build a directory entry list (BPAM)” on page 155.

September 2022 refresh

- For APAR OA62553, MAREA of “ACB—Generate an access method control block at assembly time” on page 9 macro is updated. (APAR OA62553, which also applies to z/OS V2R4.)
- Offset 145 (91) of Table 64 on page 346 is updated.

August 2022 refresh

- “GET—Retrieve a record” on page 47 is updated.

June 2022 refresh

- For APAR OA59175 (which also includes z/OS V2R3 and z/OS V2R4), updates are made to “BLDVRP—Build VSAM resource pool” on page 17.

Summary of changes for z/OS Version 2 Release 4 (V2R4)

Changes made for z/OS V2R4

New

August 2020 refresh

- Added error codes 9(X'9'), A(X'A'), and 98(X'62') to VSAM Macro Return and Reason Codes. For more information, see “Reason code (successful request)” on page 123 and “Reason code (logical errors)” on page 124.

June 2020 refresh

- New OPTCD parameter values added in “RPL—Generate a request parameter list at assembly time” on page 70, “GENCB—Generate a request parameter list at execution time” on page 40, and “MODCB—Modify a request parameter list” on page 57. New MACRF parameter values added in “ACB—Generate an access method control block at assembly time” on page 9, “GENCB—Generate an access method control block at execution time” on page 32, and “MODCB—Modify an access method control block” on page 54.
- New CONCURRENTRW parameter and values added to the DCBE macro, in “DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)” on page 230 and “Data control block extension (DCBE)” on page 393.

Prior to June 2020 refresh

- APAR OA56853 resulted in a minor change to Table 56 on page 337.

Changed

June 2021 refresh

Updates are made to Using POINT with a basic format data set, large format data set or PDS. For more information, see “POINT—Position for access (BPAM and BSAM—tape and DASD only)” on page 294.

December 2020 refresh

With APAR OA56622, updated DSENCRYPT of the DCBE macro. For more information, see “DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)” on page 230.

November 2020 refresh

With APAR OA56622, updated DSENCRYPT of the DCBE macro. For more information, see “DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)” on page 230.

September 2020 refresh

- With APAR OA56622, added DSENCRYPT to the DCBE macro and VERSION to the ISITMGD macro. For more information, see “DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)” on page 230, “ISITMGD—Is the data set system-managed? (BPAM, BSAM, QSAM)” on page 278, Appendix B, “Non-VSAM control blocks,” on page 373, “Data control block extension (DCBE)” on page 393 and “Direct access storage devices” on page 405.

Prior to June 2020 refresh

- APAR OA58692 provided updates to OPEN return code 179 (x'B3'). For more information, see “OPEN return and reason codes” on page 107.
- APAR OA53383 provided updates to the ACBERFLG, which is set to 174 (x'AE'). For more information, see “OPEN return and reason codes” on page 107.
- A section in “PUT—Write next record (QSAM)” on page 302 was rewritten for clarity.

Summary of changes for z/OS Version 2 Release 3 (V2R3)

Changes made for z/OS V2R3

New

- New reason code added for “DESERV GET_ALL_G function reason codes” on page 262.
- New example code added for “Data above the 2 GB bar” on page 150.
- New HYPERWRITE keyword added to “DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)” on page 230.
- New reason codes added to the DCBE macro, in “DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)” on page 230 and “Data control block extension (DCBE)” on page 393.

Changed

- Changes to Recovering a Generation: directory action=RECOVERG in “STOW—Update partitioned data set directory (BPAM)” on page 333.
- Changes to the 'type' keyword in “READ—Read a block (BPAM and BSAM)” on page 308, and “WRITE—Write a block (BPAM and BSAM)” on page 363.
- Changes to the ISITMGD macro in “ISITMGD—Is the data set system-managed? (BPAM, BSAM, QSAM)” on page 278.
- Changes to the CLOSE macro in “CLOSE—Disconnect program and data (BDAM, BISAM, BPAM, BSAM, QISAM, and QSAM)” on page 165.
- Changes to the VSAM macro in “Reason code (logical errors)” on page 124.
- Changes to the ACBERFLG field in “OPEN return and reason codes” on page 107.

Part 1. VSAM macro instructions

Chapter 1. Introduction to VSAM programming

You use the virtual storage access method (VSAM) to organize data and maintain information about that data in a catalog. Perform VSAM programming using access method services commands and VSAM macros.

- **Access method services**. You define VSAM data sets and establish catalogs using a multi-function services program called access method services.
- **Job control language**. You can define VSAM data sets using JCL.
- **Dynamic allocation**. You can define or allocate to data sets using dynamic allocation, which is SVC 99. Dynamic allocation is described in *z/OS MVS Programming: Authorized Assembler Services Guide*. VSAM supports the `nocapture` option of dynamic allocation. It reduces overhead of dynamic allocation and reduces virtual storage usage below the 16 MB line.
- **VSAM macro instructions**. Two types of VSAM macros are used to process VSAM data sets:
  - ***Control block macros*** generate control blocks of information needed by VSAM to process the data set.
  - ***Request macros*** are used to retrieve, update, delete, or insert logical records.

All macros described in this book are in the main system macro library, SYS1.MACLIB.

You can use 24-bit or 31-bit addressing mode for VSAM programs. If you use 31-bit support, see *z/OS DFSMS Using Data Sets* for procedures and restrictions.

Chapter 2. VSAM macro descriptions and examples

This chapter contains VSAM macro formats and examples.

The macros that work at assembly time allow you to specify subparameter values as absolute numeric expressions, character strings, codes, and expressions that generate valid relocatable A-type address constants.

The macros that work at execution allow you also to specify these values as:

- Register notation, where the expression designating a register from 2 through 12 is enclosed in parentheses. For example, (2) and (REG), where REG is a label equated to a number from 2 through 12.
- An expression of the form (S,scon), where scon is an expression valid for an S-type address constant, including the base-displacement form.
- An expression of the form (*,scon), where scon is an expression valid for an S-type address constant, including the base-displacement form, and the address specified by scon is indirect—that is, it gives the location of the area that contains the value for the subparameter.

For most programming applications, you can use register notation or absolute numeric expressions for numbers, character strings for names, and register notation or expressions that generate valid A-type address constants for addresses. “Subparameters with GENCB, MODCB, SHOWCB, and TESTCB” on page 5, gives all the ways of coding each parameter for the macros that work at execution time.

You can write a reentrant program **only** with execution-time macros. “Use of list, execute, and generate forms of VSAM macros” on page 6, describes alternative ways of coding these macros for reentrant programs. This chapter describes the standard form of these macros.

Subparameters with GENCB, MODCB, SHOWCB, and TESTCB

The addresses, names, numbers, and options required with subparameters in GENCB, MODCB, SHOWCB, and TESTCB can be expressed in a variety of ways:

- An **absolute numeric expression**, for example, STRNO=3 and COPIES=10.
- A **code or a list of codes separated by commas and enclosed in parentheses**, for example, OPTCD=KEY or OPTCD=(KEY,DIR,IN).
- A **character string**, for example, DDNAME=DATASET.
- A **register from 2 through 12 that contains an address or numeric value**, for example, SYNAD=(3); equated labels can be used to designate a register, for example, SYNAD=(ERR), where the following equate statement has been included in the program: ERR EQU 3.
- An **expression of the form (S,scon)**, where scon is an expression valid for an S-type address constant, including the base-displacement form. The contents of the base register are added to the displacement to obtain the value of the keyword. For example, if the value of the keyword being represented is a numeric value (that is, COPIES, LENGTH, RECLEN), the contents of the base register are added to the displacement to determine the numeric value. If the value of the keyword being represented is an address constant (that is, WAREA, EXLST, EODAD, ACB), the contents of the base register are added to the displacement to determine the value of the address constant.
- An **expression of the form (*,scon)**, where scon is an expression valid for an S-type address constant, including the base-displacement form. The address specified by scon is **indirect**, that is, it is the address of an area that contains the value of the keyword. The contents of the base register are added to the displacement to determine the address of the fullword of storage that contains the value of the keyword.

  If an indirect S-type address constant is used, the value it points to must meet the following criteria:

  – If it is a numeric quantity or an address, it must occupy a fullword of storage.

  - If it is an alphanumeric character string, it must occupy two words of storage, be left aligned, and be filled on the right with blanks.
- An **expression valid for a relocatable A-type address constant**, for example, AREA=MYAREA+4.

The specified keyword determines the type of expressions that can be used. Also, register and S-type address constants cannot be used when MF=L is specified.

Use of list, execute, and generate forms of VSAM macros

The BLDVRP, DLVRP, GENCB, MODCB, SHOWCB, and TESTCB macros build a parameter list describing in codes the actions shown by the subparameters you specify and pass the list to VSAM to take the suggested action.

The list, execute, and generate forms of BLDVRP, DLVRP, GENCB, MODCB, SHOWCB, and TESTCB allow you to write reentrant programs, to share parameter lists, and to modify a parameter list before using it.

Following is a brief description of the list, execute, and generate forms:

- The list form is used to build the parameter list either in line (called a ***simple list***) or in an area remote from the macro expansion (called a ***remote list***). Both the simple- and the remote-list forms allow you to build a single parameter list that can be shared.
- The execute form is used to modify a parameter list and to pass it to VSAM for action.
- The generate form is used to build the parameter list in a remote area and to pass it to VSAM for action.

The list, execute, and generate forms of the BLDVRP, DLVRP, GENCB, MODCB, SHOWCB, and TESTCB macros have the same format as the standard forms, except for:

- An additional keyword, MF.
- Keywords that are required in the standard form may be optional in the list, execute, and generate forms or may not be allowed in the execute form. The meaning of the keywords, however, and the notation that may be used to express addresses, names, numbers, and option codes are the same.

The following sections describe the format of the MF keyword and the use of list, execute, and generate forms. They also show the optional and invalid subparameters.

List-form keyword

The format of the MF keyword for the list form is:

MF={L|(L,*address*[,*label*])}

where:

L
specifies that this is the list form of the macro.

If you code MF=L, without the *address* parameter, then,

- Register notation and expressions that generate S-type address constants cannot be used
- The parameter list is built in line, which means that the program is not reentrant if the parameter list is modified at execution.

address
specifies the address of a remote area in which the macro expansion builds a parameter list. Coding the *address* parameter will result in generating executable code to initialize the remote parameter list. You can modify this parameter list with later calls to the execute form or update it with later invocations of the list form using the same *address* parameter. The area must begin on a fullword boundary and be large enough for the parameter list. You can specify the address in register notation or as an expression valid for a relocatable A-type address constant or a direct or indirect S-type address constant.

label
specifies a unique name used in an EQU instruction in the expansion of the macro. *Label* is equated to the length of the parameter list. You do not have to know the length of the parameter list if you code *label*; the expansion of the macro determines the amount of storage required.

The size, in fullwords, of a parameter list is:

- For GENCB, 4, plus 3 times the number of ACB, EXLST, or RPL keywords specified (plus 1 for DDNAME, EODAD, JRNAD, LERAD, or SYNAD)
- For MODCB, 3, plus 3 times the number of ACB, EXLST, or RPL keywords specified (plus 1 for DDNAME, EODAD, JRNAD, LERAD, or SYNAD)
- For SHOWCB, 5, plus 2 times the number of fields specified in the FIELDS keyword
- For TESTCB, 8 (plus 1 for either DDNAME, STMST, EODAD, JRNAD, LERAD, or SYNAD).

If you code MF=(L,address,label), the parameter list is built in the remote area specified. The expansion of the macro equates *label* with the length of the parameter list.

Execute-form keyword

The format of the MF keyword for the execute form is:

`MF=(E,`*address*`)`

where:

E
specifies that this is the execute form of the macro.

address
specifies the address of the parameter list.

Expansion of the execute form of the macro results in executable code that causes:

1. A parameter list to be modified, if requested
2. Control to be passed to a routine that satisfies the request.

You may not use the execute form to add an entry to a parameter list. If you try to add an entry, you receive a return code of 8 in register 15.

Generate-form keyword

The format of the MF keyword for the generate form is:

`MF=(G,`*address*`[,`*label*`])`

where:

G
specifies that this is the generate form of the macro.

address
specifies the address of a remote area in which the parameter list is to be built. The area must begin on a fullword boundary.

label
specifies a unique name that is used in an EQU instruction in the expansion of the macro. *Label* is equated to the length of the parameter list. You do not have to know the length of the parameter list if you code *label*; the expansion of the macro determines the amount of storage required.

If you code MF=(G,address), the parameter list is built in the remote area specified.

If you code MF=(G,address,label), the parameter list is built in the remote area specified. The expansion of the macro equates the length of the parameter list to *label*.

Examples of generate, list, and execute forms

Table 1 on page 8 shows which forms of GENCB, MODCB, SHOWCB, and TESTCB should be used in reentrant/nonreentrant and shared/nonshared environments.

Table 1. Reentrant Programming

	Reentrant	**Nonreentrant**
Shared	MF=(L,*address*[,*label*])	MF=L
	MF=(E,*address*)	MF=(E,*address*)
Nonshared	MF=(G,*address*[,*label*])	Standard Form

The figure shows that:

- To share parameter lists in a reentrant program, the remote-list form should be used with the execute form.
- To share parameter lists in a nonreentrant program, the simple-list form should be used with the execute form.
- If you do not intend to share parameter lists, the generate form should be used in reentrant programs and the standard form should be used for nonreentrant programs.

The following examples show how the generate, list, and execute forms work.

Example: generate form (reentrant)

In this example, the generate form of GENCB is used to create a default request parameter list (RPL) in a reentrant environment.

```
        LA      10,LEN1          Get length of the parameter list.

        GETMAIN R,LV=(10)        Get storage for the area in which      x
                                 the parameter list is to be built.     x
        LR      2,1              Save address of parameter-list area.

        GENCB   BLK=RPL,                                                x
                MF=(G,(2),LEN1)
```

The macro expansion equates LEN1 to the length of the parameter list, as follows:

```
+LEN1 EQU 16
```

The parameter list is built in the area acquired by the GETMAIN macro and pointed to by register 2. This list is used by VSAM to build the RPL. VSAM returns the RPL address in register 1 and the RPL length in register 0. If the WAREA and LENGTH parameters are used, the RPL is built at the WAREA address.

Example: remote-list form (reentrant)

In this example, the remote-list form of MODCB is used to build a parameter list that will later be used to modify the MACRF bits in the access method control block ANYACB.

```
        LA      8,LEN2           Get length of the parameter list.

        GETMAIN R,LV=(8)         Get storage for the area in which the  x
                                 parameter list is to be built.
        LR      3,1              Save address of the parameter-list area.

        MODCB   ACB=ANYACB,                                             x
                MACRMF=SEQ,MF=(L,(3),LEN2)
```

The macro expansion equates the length of the parameter list to LEN2, as follows:

```
+LEN2 EQU 24
```

This parameter list is built in the remote area pointed to by register 3. The list is used by VSAM to modify the ACB when an execute form of MODCB is issued (see next example). The list form only creates a parameter list; it does not modify the ACB.

Example: execute form (reentrant)

In this example, the execute form of MODCB is used to modify the address of the access method control block and MACRF codes in the parameter list created by the remote-list form of MODCB in the previous example.

```
MODCB  ACB=MYACB,MACRF=(ADR,SEQ,OUT),MF=(E,(3))
```

The parameter list pointed to by register 3 is changed so that the ACB and MACRF parameter values in the execute form override those in the list form. The access method control block, MYACB, is then modified to MACRF=(ADR,SEQ,OUT).

The access method control block at ANYACB is not changed by either of these examples.

ACB—Generate an access method control block at assembly time

Use the ACB macro to generate an access method control block at assembly time.

The format of the ACB macro is:

[*label*]	ACB	[AM=VSAM] [,BSTRNO=*abs expression*] [,BUFND=*abs expression*] [,BUFNI=*abs expression*] [,BUFSP=*abs expression*] [,DDNAME=*character string*] [,EXLST=*address*] [,MACRF=([ADR][,CNV][,KEY] [,CFX|NFX] [,DDN|DSN] [,DFR|NDF] [,DIR][,SEQ][,SKP] [,ICI|NCI] [,IN][,OUT] [,LEW|NLW] [,NIS|SIS] [,NRM|AIX] [,NRS|RST] [,NSR|LSR|GSR|RLS] [,NUB|UBF])] [,DB={YES|NO}] [,MAREA=*address*] [,MLEN=*abs expression*] [,PASSWD=*address*] [,RLSREAD={NRI|CR|NORD}] [,RMODE31={ALL|BUFF|CB|NONE}] [,SHRPOOL={0|*abs expression*}] [,STRNO=*abs expression*] [,ZHYPERWRITE={YES|NO}]

Values for ACB macro subparameters can be specified as absolute numeric expressions, character strings, codes, and expressions that generate valid relocatable A-type address constants.

label
specifies 1 to 8 characters that provide a symbolic address for the access method control block that is assembled. If you omit the DDNAME parameter, *label* serves as the ddname.

AM=VSAM
specifies that the access method using this control block is VSAM.

BSTRNO=*abs expression*
specifies the number of strings that are initially allocated for access to the base cluster of a path. BSTRNO must be a number between 0 and 255. The default is STRNO. BSTRNO is ignored if the object being opened is not a path. If the number that is specified for BSTRNO is insufficient, VSAM dynamically extends the number of strings as needed for access to the base cluster.

BSTRNO can influence performance. The VSAM control blocks for the set of strings that is specified by BSTRNO are allocated in contiguous virtual storage. This is not guaranteed for the strings allocated by dynamic extension.

This parameter is only applicable to MACRF=NSR.

This parameter has no effect for z/OS UNIX files. This is the case when an application program uses the VSAM interface to access an z/OS UNIX file.

BUFND=*abs expression*
specifies the number of I/O buffers that VSAM is to use for transmitting data between virtual and auxiliary storage. A buffer is the size of a control interval in the data component. BUFND must be a number between 0 and 32767. The minimum number that you can specify is 1 plus the number that is specified for STRNO. (If you omit STRNO, BUFND must be at least 2, because the default for STRNO is 1.) The number can be supplied through the JCL DD AMP parameter and through the macro. The default is the minimum number that is required. The minimum buffer specification does not provide optimum sequential processing performance. Generally, the more data buffers that are specified, the better the performance.

Additional data buffers benefit direct inserts or updates during control area splits and benefit spanned record accessing. See *z/OS DFSMS Using Data Sets* for more information on optimizing performance and system-managed buffering.

This parameter is applicable only to MACRF=NSR; it is ignored when MACRF=RLS is specified.

This parameter has no effect for z/OS UNIX files.

BUFNI=*abs expression*
specifies the number of I/O buffers that VSAM is to use for transmitting the contents of index entries between virtual and auxiliary storage for keyed access. A buffer is the size of a control interval in the index. BUFNI must be a number between 0 and 32767. The minimum number is the number that is specified for STRNO (if you omit STRNO, BUFNI must be at least 1, because the default for STRNO is 1). You can supply the number through the JCL DD AMP parameter and through the macro. The default is the minimum number that is required.

Additional index buffers improve performance by providing for the residency of some or all of the high-level index, thereby minimizing the number of high-level index records retrieved from DASD for key-direct processing. For more information on optimizing performance, see *z/OS DFSMS Using Data Sets*.

The default is the minimum number that is required.

This parameter is only applicable to MACRF=NSR.

This parameter has no effect for z/OS UNIX files.

BUFSP=*abs expression*
specifies the maximum number of bytes of virtual storage to be used for the data and index I/O buffers. VSAM gets the storage in your program's address space. If you specify less than the amount of space that was specified in the BUFFERSPACE parameter of the DEFINE command when the data set was defined, VSAM overrides your BUFSP specification upward to the value that is specified in BUFFERSPACE. (BUFFERSPACE, by definition, is the least amount of virtual storage that is ever

provided for I/O buffers.) However, if BUFSP is specified and the amount specified is much too small — smaller than the minimum amount of buffer storage required to process the data set — VSAM cannot open the data set. The minimum amount is described under BUFND and BUFNI, above.

You can supply BUFSP through the JCL DD AMP parameter and through the macro. If you do not specify BUFSP in either place, the amount of storage that is used for buffer allocation is the *largest* of the following amounts:

- Amount that is specified in the catalog (BUFFERSPACE)
- Amount that is determined from BUFND and BUFNI or
- Minimum storage that is required to process the data set with its specified processing options

A valid BUFSP amount takes precedence over the amount that is called for by BUFND and BUFNI. If the BUFSP amount is greater than the amount that is called for by BUFND and BUFNI, the extra space is allocated under the following conditions:

- When MACRF indicates direct access only, additional index buffers are allocated.
- When MACRF indicates sequential access, one additional index buffer and as many data buffers as possible are allocated.

If the BUFSP amount is less than the amount that is called for by BUFND and BUFNI, the number of data and index buffers is decreased under the following conditions:

- When MACRF indicates direct access only, the number of data buffers is decreased to not fewer than the minimum number. Then, if required, the number of index buffers is decreased until the amount that is called for by BUFND and BUFNI complies with the BUFSP amount.
- When MACRF indicates sequential access, the number of index buffers is decreased to not fewer than 1 more than the minimum number. Then, if required, the number of data buffers is decreased to not fewer than the minimum number. If still required, 1 more is subtracted from the number of index buffers.
- Neither the number of data buffers nor the number of index buffers is decreased to fewer than the minimum number.

If the index does not exist or is not being opened, only BUFND, and not BUFNI, enters these calculations.

The BUFFERSPACE must not exceed 16776704.

This parameter is only applicable to MACRF=NSR.

This parameter has no effect for z/OS UNIX files.

DDNAME=*character string*
specifies 1 to 8 characters that identify the data set you want to process by specifying the JCL DD statement for the data set. You may omit DDNAME and provide it through the label or through the MODCB macro before opening the data set. MODCB is described in "MODCB—Modify an access method control block" on page 54.

EXLST=*address*
specifies the address of a list of addresses of exit routines that you are providing. The list must be established by the EXLST or GENCB macro. If you use the EXLST macro, you can specify its label here as the address of the exit list. If you use GENCB, you can specify the address returned by GENCB in register 1 or the label of an area you supplied to GENCB for the exit list.

To use the exit list, you must code this EXLST parameter. Omitting this parameter means that you have no exit routines. Exit routines are described in *z/OS DFSMS Using Data Sets*.

MACRF=([ADR][,CNV][,KEY]
[,CFX|NFX]
[,DDN|DSN]
[,DFR|NDF]
[,DIR][,SEQ][,SKP]
[,ICI|NCI]
[,IN][,OUT]
[,LEW|NLW]
[,NIS|SIS]
[,NRM|AIX]
[,NRS|RST]
[,NSR|LSR|GSR|RLS]
[,NUB|UBF])

specifies the kinds of processing you will do with the data set. The subparameters must be significant for the data set. For example, if you specify keyed access for an entry-sequenced data set (ESDS), you cannot open the data set. You must specify all the types of access you are going to use, whether you use them concurrently or by switching from one to the other. Table 2 on page 12 gives the subparameters. Each group of subparameters has a default value (shown by underlining). You may specify subparameters in any order. You may specify both ADR and KEY to process a key-sequenced data set (KSDS). You may specify both DIR and SEQ; with keyed access, you may specify SKP as well. If you specify OUT and want merely to retrieve some records and also update, delete, or insert others, you need not also specify IN.

Table 2. MACRF Options

Option	Meaning
ADR	Addressed access to a key-sequenced or entry-sequenced data set; RBAs are used as search arguments and sequential access is by entry sequence. VSAM RLS does not support ADR access to a KSDS.
CNV	Access is to the entire contents of a control interval rather than to an individual data record. If the data set is password protected, you must supply the address of the control or higher-level password in the ACB PASSWD parameter. Recommendation: Use RACF® or a functionally equivalent program instead of VSAM passwords. For VSAM RLS and z/OS UNIX files, CNV is invalid. If it is specified for a z/OS UNIX file, it results in an OPEN failure.
KEY	Keyed access to a relative record data set (RRDS) or key-sequenced data set. Keys or relative record numbers are used as search arguments and sequential access is by key or relative record number. KEY processing is not affected by RLS.
CFX	If you use ICI, OPEN fixes control blocks and I/O buffers and they remain fixed until the ACB is closed. For RLS and z/OS UNIX files, this subparameter has no effect.
NFX	OPEN does not fix control blocks or I/O buffers. VSAM fixes and unfixes pages dynamically as needed. For RLS and z/OS UNIX files, NFX is assumed.
DDN	Subtask shared control block connection is based on common ddnames. For RLS and z/OS UNIX files, this subparameter has no effect.
DSN	Subtask shared control block connection is based on common data set names. For RLS and z/OS UNIX files, this subparameter has no effect.

Table 2. MACRF Options (continued)

Option	Meaning
DFR	With shared resources, writes for direct PUT requests are deferred until the WRTBFR macro is issued or until VSAM needs a buffer to satisfy a GET request. Deferring writes saves I/O requests in cases where subsequent requests can be satisfied by the data already in the buffer pool. For RLS, DFR is ignored and direct request modified buffers are immediately written to disk and the CF (coupling facility). This subparameter has no effect for z/OS UNIX files.
NDF	Writes are not deferred for direct PUTs. For RLS, NDF is ignored and direct request modified buffers are immediately written to disk and the CF (coupling facility).
DIR	Direct access to an RRDS, KSDS, or ESDS.
SEQ	Sequential access to an RRDS, KSDS, or ESDS.
SKP	Skip-sequential access to an RRDS or KSDS. Used only with keyed access in a forward direction.
ICI	Processing is limited to improved control interval processing; access is faster because fewer processor instructions are executed. ICI processing is not allowed for extended format data sets. For RLS and z/OS UNIX files, this subparameter has no effect.
NCI	Processing other than improved control interval processing.
IN	Retrieval of records of a RRDS, KSDS, or ESDS; (not allowed for an empty data set). If the data set is password protected, you must supply the address of the read or higher-level password in the ACB PASSWD parameter.
OUT	Storage of new records in a RRDS, KSDS, or ESDS (not allowed with addressed access to a KSDS). Update of records in a RRDS, KSDS, or ESDS. Deletion of records from a RRDS or KSDS. If the data set is password protected, you must supply the address of the update or higher-level password in the ACB PASSWD parameter.
LEW	Using LSR, if an exclusive control conflict is encountered, VSAM defers the request until the resource becomes available.
NLW	With this value specified, instead of deferring the request, VSAM returns the exclusive control return code 20 (X'14') to the application program. The application program is then able to determine the next action.
NIS	Normal insert strategy. This subparameter has no effect for z/OS UNIX files.
SIS	Sequential insert strategy (split control intervals and control areas at the insert point rather than at the midpoint when doing direct PUTs); although positioning is lost and writes are done after each direct PUT request, SIS allows more efficient space usage when direct inserts are clustered around certain keys. This subparameter has no effect for z/OS UNIX files.
NRM	The object to be processed is the one named in the specified ddname.
AIX	The object to be processed is the alternate index of the path specified by ddname, rather than the base cluster though the alternate index. For RLS, the AIX subparameter is invalid. This subparameter has no effect for z/OS UNIX files.
NRS	Data set is not reusable.

Table 2. MACRF Options (continued)

Option	Meaning
RST	Data set is reusable (high-used RBA is reset to 0 during OPEN). If the data set is password protected, you must supply the address of the update or higher-level password in the ACB PASSWD parameter.
NSR	Nonshared resources.
LSR	Local shared resources. Each address space may have up to 256 index resource pools and 256 data resource pools independent of other address spaces. Unless you are using the default, SHRPOOL=0, you must specify the SHRPOOL parameter to indicate which resource pool you are using. Specifying LSR causes a data set to use the local resource pool built by the BLDVRP macro. If an index resource pool exists at the time an OPEN macro is issued, the index for a KSDS is connected to the index resource pool. This parameter is invalid for z/OS UNIX files and if specified results in an open failure.
GSR	Global shared resources; all address spaces may have local and global resources pools, where tasks in an address space with a local resource pool may use either the local resource pool or the global resource pool. This parameter is invalid for compressed format data sets. If specified for a z/OS UNIX file, it results in an open failure.
RLS	RLS specifies that VSAM record level sharing protocols are used. RLS and NSR/LSR/GSR are mutually exclusive. RLS implies that VSAM uses cross system record level locking as opposed to CI locking, uses CF for buffer consistency, and manages a system wide local cache. When you specify this parameter, OPEN will fail for: • Linear data sets • ADR access to a KSDS • CNV access to any data set organization • Data sets defined with imbedded indexes • z/OS UNIX files.
NUB	Management of I/O buffers is left up to VSAM. For RLS, you must specify NUB.
UBF	Management of I/O buffers is left up to the user. The work area specified by the RPL (or GENCB) AREA parameter is the I/O buffer. VSAM transmits the contents of a control interval directly between the work area and direct access storage. UBF is valid when OPTCD=MVE and MACRF=CNV are specified. When ICI is specified, UBF is assumed. For RLS, UBF is invalid.

DB={YES|NO}

YES
specifies that RLS processes the VSAM database (VSAMDB) as a database, honoring the DATABASE specification in DEFINE CLUSTER or DEFINE AIX, not as a regular KSDS. YES is the default.

NO
specifies that RLS processes the VSAM database (VSAMDB) as a regular KSDS, not as a database, ignoring the DATABASE specification in DEFINE CLUSTER or DEFINE AIX.

Note: Non-RLS VSAM always treats a VSAM database as a regular KSDS, ignoring DB={YES|NO} and the DATABASE specification in DEFINE CLUSTER or DEFINE AIX.

MAREA=*address*
specifies the address of an optional OPEN/CLOSE or TYPE=T option (CLOSE macro) message area. For non-RLS VSAM, see "OPEN/CLOSE message area for multiple reason or attention messages" on page 116 for more information. ('MAREA is ignored for RLS' is deleted.)

For VSAM RLS, when an OPEN error occurs, the area pointed to by MAREA contains 4 bytes of error information, in this order: a 1-byte OPEN error code; 2-byte function code; and the 1-byte reason code as found in the ACBERFLG field of the ACB.

MLEN=*abs expression*
specifies the length of an optional OPEN/CLOSE or TYPE=T option (CLOSE macro) message area. The default is 0. The maximum length is 32KB. See "OPEN/CLOSE message area for multiple reason or attention messages" on page 116 for more information. MLEN is ignored for RLS.

PASSWD=*address*
specifies the address of a field containing the highest-level password required for the types of access indicated by the MACRF parameter. The first byte of the field pointed to contains the length (in binary) of the password (maximum of 8 bytes). Zero indicates that no password is supplied. If the data set is password protected and you do not supply a required password in the access method control block, VSAM gives the console operator the opportunity to supply it when you open the data set.

Data sets which are opened for RLS processing must be SMS-managed data sets which have had password processing ignored.

This parameter has no effect for z/OS UNIX files.

RLSREAD={NRI|CR|NORD}
RLSREAD (for RLS) specifies the read integrity options that apply to GET requests that are issued against this ACB. This parameter overrides the read integrity options that are specified in the RLS JCL parameter. You can override the RLSREAD parameters for a specific GET request by specifying the read integrity options in the RPL OPTCD parameter.

NRI
specifies no read integrity. NRI is a performance option. When you specify NRI, VSAM does not obtain a lock on the record.

CR
specifies consistent read integrity. CR ensures that only records that have been committed are read.

NORD
specifies that the read integrity option that is used is determined either by the RLS JCL specification or by options that are specified on the GET request.

For access modes other than RLS, RLSREAD is ignored.

RMODE31=[ALL|BUFF|CB|NONE]
specifies where VSAM OPEN obtains virtual storage (above or below 16 megabytes) for control blocks and I/O buffers.

The values specified by the RMDE31 parameter have an effect only before issuing an OPEN. At all other times, changing these values has no effect on the residency of the control blocks and I/O buffers.

If MACRF=RLS is specified, RMODE31=ALL is assumed. For RLS and DFSMStvs, VSAM control blocks and buffers are located in a data space owned by the SMSVSAM server address space and are not directly addressable.

RMODE31= can also be specified on the JCL AMP parameter.

ALL
specifies that both VSAM control blocks and I/O buffers are obtained above 16 megabytes.

BUFF
specifies that only VSAM I/O buffers are obtained above 16 megabytes.

CB
specifies that only VSAM control blocks are obtained above 16 megabytes.

NONE
specifies that both I/O buffers and VSAM control blocks are built below 16 megabytes. This is the default.

SHRPOOL={*abs expression*|0}
specifies which LSR pool is connected to the ACB. This parameter is valid only when MACRF=LSR is also specified. SHRPOOL must be a number between 0 and 255. The default is 0.

STRNO=*abs expression*
specifies the number of requests requiring concurrent data set positioning VSAM is prepared to handle. STRNO must be a number between 1 and 255. The default is 1. A request is defined by a given request parameter list or chain of request parameter lists. The string number is equal to the number of requests issued concurrently for all the data sets sharing the resource pool. See "RPL—Generate a request parameter list at assembly time" on page 70 and "GENCB—Generate a request parameter list at execution time" on page 40 for information on request parameter lists. When records are loaded into an empty data set, the STRNO value in the access method control block must be 1.

VSAM dynamically extends the number of strings as they are needed by concurrent requests for this ACB. This automatic extension can influence performance. The VSAM control blocks for the set of strings specified by STRNO are allocated on contiguous virtual storage, but this is not guaranteed for the strings allocated by dynamic extension. Dynamic string addition cannot be done when using the following options:

- Load mode
- ICI
- LSR or GSR

For STRNO, you should specify the total number of request parameter lists or chains of request parameter lists that you are using to define requests. (VSAM needs to remember only one position for a chain of request parameter lists.) However, each position beyond the minimum number that VSAM needs to be able to remember requires additional virtual storage space for these parameters:

- A minimum of one data I/O buffer and, for keyed access, one index I/O buffer (the size of an I/O buffer is the control interval size of a data set)
- Internal control blocks and other areas

For RLS, STRNO is ignored. Strings are dynamically acquired up to a limit of 1024.

STRNO >1 is not supported for z/OS UNIX files. If you specify a value greater than 1, OPEN fails.

ZHYPERWRITE={YES|NO}
specifies YES to enable zHyperWrite support for the data set.

Example 1: ACB macro

In this example, the ACB macro is used to identify a data set to be opened and to specify the types of processing to be performed. The access method control block generated by this example is built when the program is assembled.

```
BLOCK    ACB   AM=VSAM,BUFND=4,          BLOCK gives symbolic          x
               BUFNI=3,                  address of the access         x
               BUFSP=19456,              method control block.         x
               DDNAME=DATASETS,                                        x
               EXLST=EXITS,                                            x
               MACRF=(KEY,DIR,SEQ,OUT),                                x
               STRNO=2,
```

The ACB macro's parameters are:

- BUFND specifies four I/O buffers for data. BUFNI specifies three I/O buffers for index entries. BUFSP specifies 19456 bytes of buffer space, enough space to accommodate control intervals of data that are 4096 bytes and control intervals of index entries that are 1024 bytes.
- DDNAME specifies this access method control block is associated with a DD statement named DATASETS.
- EXLST specifies the exit list associated with this access method control block is named EXITS.
- MACRF specifies keyed-direct and keyed-sequential processing for both insertion and update.

- STRNO specifies two requests will require concurrent positioning.
- Since the type of resources are not specified, NSR is assumed.

Example 2: ACB macro

In this example, the ACB macro is used to identify a data set to be opened and to specify the types of processing to be performed. The access method control block generated by this example is built when the program is assembled. The caller requests that the VSAM control blocks and I/O buffers be obtained above 16 megabytes, if possible.

```
BLOCK2   ACB   AM=VSAM,                    BLOCK2 gives symbolic        x
               DDNAME=DATASETS,            address of the access        x
               EXLST=EXITS,                method control block.        x
               MACRF=(KEY,DIR,SEQ,OUT),                                 x
               RMODE31=ALL
```

The ACB macro's parameters are:

- DDNAME specifies this access method control block is associated with a DD statement named DATASETS.
- EXLST specifies the exit list associated with this access method control block is named EXITS.
- MACRF specifies keyed-direct and keyed-sequential processing for both insertion and update.
- RMODE31=ALL specifies both VSAM control blocks and buffers may reside above 16 megabytes.
- Since the type of resources are not specified, NSR is assumed.

BLDVRP—Build VSAM resource pool

Use the BLDVRP macro to build a VSAM resource pool.

The format of the BLDVRP macro is:

[*label*]	BLDVRP	BUFFERS=(*size*(*abs expression*[,*Hiperspace*]), *size*(*abs expression*[,*Hiperspace*]),...) [,FIX={BFR|IOB|(BFR,IOB)}] [,KEYLEN=*length*] [,MODE={24|31}] [,RMODE31={ALL|BUFF|CB|NONE}] [,SHRPOOL={0|*abs expression*}] ,STRNO=*abs expression* [,TYPE={LSR|(LSR,DATA|INDEX)|GSR}]

The BLDVRP macro has a standard form and list and execute forms. The standard form builds a parameter list and passes control to VSAM to build the resource pool. The list and execute forms are described in “Use of list, execute, and generate forms of VSAM macros” on page 6.

label
specifies 1 to 8 characters that provide a symbolic address for the BLDVRP macro.

BUFFERS=(*size*(*abs expression*[,*Hiperspace*]), *size*(*abs expression*[,*Hiperspace*]),...)
specifies the size and number of virtual and Hiperspace buffers in each buffer pool in the resource pool. The number of buffer pools in the resource pool is implied by the number of "size(abs expression,Hiperspace)" groups you specify.

The request for the virtual storage is granted even if the request for Hiperspace buffers cannot be completely fulfilled. Some specifications may have Hiperspace buffers allocated while other specifications in the same BLDVRP request may not.

When you process a KSDS, the index component and the data component share the buffers of a buffer pool. When you use an alternate index to process a base cluster, the components of the alternate

index and the base cluster share buffers. The components of alternate indexes in an upgrade set share buffers. Buffers of the appropriate size and number must be provided for all components. Each component uses the buffer pool with buffers of either the required size or larger.

LSR/GSR users: To ensure that the buffer pool built by BLDVRP is used, use the access method services DEFINE CLUSTER command to define explicitly the matching data and index control interval sizes. Hiperspace buffer sizes must match the control interval size of the data set components.

size
specifies an integer multiple of 512 or 2048 up to a maximum of 32768 bytes, where n is a positive integer from 1 to 16.

CISZ=(n x 512) or (n x 2048)

Requirement: If you specify Hiperspace buffering (*Hiperspace*), the size must be a multiple of 4096 and match the CISIZE of the data set components.

abs expression
specifies a minimum of 3 up to a maximum of 65535.

Hiperspace
specifies the number of Hiperspace buffers in the buffer pool. The default is 0. The maximum value is 16777215. Specifying many Hiperspace buffers may create virtual storage constraint problems since an 8-byte hash table entry is built in virtual address space for each Hiperspace buffer.

The Hiperspace option is ignored when TYPE=GSR is specified.

FIX={BFR|IOB|(BFR,IOB)}
specifies that I/O buffers (BFR), I/O-related control blocks (IOB), or both, are fixed in real storage. With GSR, IOB includes channel programs. If a program issues BLDVRP and specifies FIX but the program is not authorized to fix areas in real storage, FIX is ignored. To be authorized, a program must either be in supervisor state with protection key 0 to 7, or be link-edited with APF authorization. See *z/OS MVS Programming: Authorized Assembler Services Guide* for a description of the authorized program facility.

If FIX=IOB is specified for BLDVRP TYPE=INDEX, it is ignored; the FIX=IOB specified for BLDVRP TYPE=DATA is used instead.

Requirement: If FIX is specified, DLVRP must be issued by the same task that issues BLDVRP.

KEYLEN=*length*
specifies the maximum key length of the data sets that share the resource pool. The default is 255.

Requirement: If your keys are smaller than 255 bytes, specifying the exact key length saves storage space. You must provide lengths for the prime key of each KSDS and for the alternate key of each alternate index that is either used for processing or is being upgraded. Specify 0 if none of the data sets are keyed.

If KEYLEN is specified for BLDVRP TYPE=INDEX, it is ignored; the KEYLEN specified for BLDVRP TYPE=DATA is used.

RMODE31={ALL|BUFF|CB|NONE}
specifies the storage residence of the buffers and I/O related control blocks of the LSR pool identified with the SHRPOOL keyword. The RMODE31 parameter tells VSAM OPEN routines where to obtain storage for the I/O related control blocks and I/O buffers.

If RMODE31 is specified for BLDVRP TYPE=INDEX, it affects the residence of the I/O buffers but is ignored for I/O related control blocks. If RMODE31 is specified for BLDVRP TYPE=INDEX, the RMODE31 specified for BLDVRP TYPE=DATA is used to set these control blocks instead.

The RMODE31 parameter is valid only when TYPE=LSR is specified.

ALL
specifies both I/O buffers and the VSAM I/O related control blocks associated with the pool reside above 16 megabytes.

BUFF
specifies that only I/O buffers reside above 16 megabytes. This parameter is the same as the LOC=ANY parameter in previous releases.

CB
specifies only the VSAM I/O related control blocks associated with the pool reside above 16 megabytes.

NONE
specifies both I/O buffers and the VSAM I/O related control blocks associated with the pool reside below 16 megabytes. This is the default.

In previous releases, the LOC=(BELOW|ANY) parameter was used to specify that buffers in the pool be created above 16 megabytes. The RMODE31 parameter replaces the LOC parameter and the two parameters are mutually exclusive. If both are specified on the BLDVRP macro, the LOC parameter is ignored.

SHRPOOL={0|*abs expression*}
specifies the identification number of a shared resource pool. This parameter is valid only when TYPE=LSR and RMODE31 are also specified or defaulted.

0
specifies the shared pool with the ID of 0.

abs expression
specifies the shared pool with the ID of *number* where *number* can be 0 to 255. The LSR control block and buffer pool residence is determined by the RMODE31 parameter.

MODE={24|31}
specifies the format of the BLDVRP parameter list to be generated.

24
specifies that a standard form (24-bit) parameter list address be generated.

31
specifies that a long form (31-bit) parameter list address be generated. This value is required if the parameter list resides above 16 megabytes.

STRNO=*abs expression*
specifies the total number of place holders required for all the data sets sharing the resource pool. 1 is minimum; 255 is maximum.

The number should equal the potential number of requests that may be issued concurrently for all the data sets sharing the resource pool. If a request fails because of an insufficient number of place holders (you receive reason code X'40' in the RPL feedback area), you may retry the request. It is assigned a place holder if one has been released. See Table 17 on page 124 for a description of reason code X'40'.

STRNO is required for TYPE=DATA. For BLDVRP TYPE=INDEX, STRNO is not required and, if specified, is ignored. The STRNO specified by BLDVRP TYPE=DATA is used.

TYPE={LSR|(LSR,DATA|INDEX)|GSR}
specifies whether a local (LSR) or a global (GSR) resource pool is built.

LSR
specifies a local shared resource pool. A maximum of 256 data and 256 index resource pools can be built in one address space. Each resource pool must be built individually.

DATA
specifies that a data resource pool be built. LSR must also be specified or defaulted, and this resource pool must exist before an index pool with the same shared pool ID can be built.

INDEX
specifies an index resource pool be built. LSR must also be specified or defaulted. INDEX must be specified to create a separate index resource pool. If it is not specified, both data and index components use the data pools. A data pool must already exist before an index pool with the same shared pool ID can be built.

For BLDVRP TYPE=INDEX, the following parameters are ignored:

FIX=IOB
KEYLEN
RMODE31 (as it affects the setting of the I/O related control blocks)
STRNO

The FIX=IOB, KEYLEN, RMODE31, and STRNO parameters specified for BLDVRP=DATA are used instead. For example:

```
BLDVRP   TYPE=INDEX,                                                  x
         FIX=IOB,                                                     x
         KEYLEN=4,                                                    x
         RMODE31=ALL,                                                 x
         STRNO=10
```

results in the FIX, KEYLEN, and STRNO parameters being reset to the values specified in BLDVRP TYPE=DATA. The buffer pools reside above 16 megabytes but the control blocks are at the residence specified by BLDVRP TYPE=DATA.

GSR
specifies a global shared resource pool.

Only one BLDVRP TYPE=GSR may be issued for the system for each of the protection keys 0 through 7. The program that issues BLDVRP TYPE=GSR must be in supervisor state with protection key 0 to 7. For more information, see Building a Resource Pool: BLDVRP in *z/OS DFSMS Using Data Sets*.

Example 1: obtaining an LSR pool above 16 megabytes

This example shows how both a local shared resource pool and a BLDVRP parameter list residing above 16 megabytes are obtained.

```
POOL1  BLDVRP  BUFFERS=(1024(5)),                                     x
               STRNO=4,                                               x
               TYPE=LSR,                                              x
               MODE=31,                                               x
               RMODE31=ALL
```

The BLDVRP parameters are:

- BUFFERS specifies there is one buffer pool in the resource pool. This buffer pool contains 5 buffers, and each of these buffers is 1024 bytes.
- STRNO specifies 4 place holders are required for the data sets to share the resource pool.
- TYPE specifies a local resource pool is built.
- MODE specifies a parameter list is generated that may reside above or below 16 megabytes. The value of 31 must be coded if the parameter list resides above 16 megabytes.
- RMODE31 specifies the location in storage for the I/O buffers and I/O related control blocks of the LSR pool.

To connect the LSR pool to the data set, you must code the LSR and SHRPOOL parameters on the ACB. See “ACB—Generate an access method control block at assembly time” on page 9.

Example 2: request for separate data and index resource pools

This example shows how the two separate data and index resource pools with an identification equal to 3 are created.

```
POOL1  BLDVRP  BUFFERS=(2048(4)),                                     x
               TYPE=(LSR,DATA),                                       x
               SHRPOOL=3,                                             x
               STRNO=2,                                               x
               RMODE31=ALL
```

```
*
       LTR     R15,R15             Check return code.

       BNZ     ERROR               Do not build index if error.
*

POOL2  BLDVRP  BUFFERS=(1024(5)),                                    x
               TYPE=(LSR,INDEX),                                     x
               SHRPOOL=3,                                            x
               STRNO=2,                                              x
               RMODE31=ALL
```

Requirement: POOL1 must be created first because the data pool must exist before the index pool with the same shared pool ID can be built. Also, only one data and one index pool can be built for a shared pool ID.

BLDVRP—List form

The format of the list form of BLDVRP is:

[*label*]	BLDVRP	BUFFERS=(*size*(*abs expression*[,*Hiperspace*]), *size*(*abs expression*[,*Hiperspace*]),...) ,MF=L [,FIX={BFR|IOB|(BFR,IOB)}] [,KEYLEN=*length*] [,RMODE31={ALL|BUFF|CB|NONE}] [,SHRPOOL={0|n}] [,MODE={24|31}] [,STRNO=*abs expression*] [,TYPE={LSR|(LSR,DATA|INDEX)|GSR}] [,DBA={YES|NO}]

Requirement: If FIX is specified, DLVRP must be issued by the same task that issues BLDVRP. STRNO is optional in the list form of BLDVRP. If STRNO is not specified in the list form, it must be specified in the execute form. The DBA parameter specifies if VSAM is allowed to dynamically add buffers to the LSR pool if needed. It is not valid for GSR pools. YES is the default.

BLDVRP—Execute form

The format of the execute form of BLDVRP is:

[*label*]	BLDVRP	MF=(E,*address*) [,KEYLEN=*length*] [,RMODE31={ALL|BUFF|CB|NONE}] [,SHRPOOL=*abs expression*] [,MODE={24|31}] [,STRNO=*abs expression*] [,TYPE={LSR|(LSR,DATA|INDEX)|GSR}] [,DBA={YES|NO}]

The address is the address of the parameter list built by a list form of BLDVRP. If you use register notation, you may use register 1, and a register between 2 and 12. Register 1 is used to pass the parameter list to VSAM. BUFFERS may not be specified in the execute form of BLDVRP, because this parameter affects the length of the parameter list.

If MODE=31 was specified on the list form, MODE=31 must be specified on the execute form. The same is true for MODE=24.

Of the execute-form BLDVRP parameters listed above, the RMODE31 (or LOC) specification does not need to be given again on the execute form if it is specified on the list form. All of the other parameters must be specified again in the execute form if they are specified on the list form. Otherwise, their default values override the values specified on the list form.

CHECK—Wait for completion of a request

Use the CHECK macro to wait for completion of an I/O request.

The format of the CHECK macro is:

[*label*]	CHECK	RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the CHECK macro.

RPL=*address*
specifies the address of the request parameter list that defines the request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Example 1: check return codes after an asynchronous request

In this example, return codes are checked after an asynchronous request. The CHECK macro is used to cause an exit to be taken if there is a logical or physical error or if the end of the data set is reached.

```
REQPARMS RPL   OPTCD=ASY
         ...
         GET   RPL=REQPARMS
         LTR   15,15           Was the request completed successfully?
         BNZ   REJECTED        Zero means the request was accepted.       x
                               If not accepted, register 15 contains      x
                               4: REQPARMS is active for another          x
                               request.  Continue working on something    x
                               not dependent on the request.

         CHECK RPL=REQPARMS    CHECK would cause one of the three         x
                               exits to be taken if there was a logi-     x
                               cal or physical error or if the end of     x
                               the data set was reached and an active     x
                               exit list exists.

         LTR   15,15           Test return indication is register 15.
```

```
         BNZ   FAILURE         Zero means the request completed           x
                               successfully.  If it failed, register      x
                               15 contains 8 or 12:  there was            x
                               a logical or a physical error.
         ...
```

```
REJECTED ...

FAILURE  ...
```

Unless you provide exit routines that terminate processing, always test register 15 after the CHECK. If a routine returns to VSAM, register 15 is reset and control is passed back to your program immediately after the CHECK. An error analysis routine normally issues SHOWCB or TESTCB to examine the feedback field in the request parameter list, so that, when your processing program gets control back, it does not have to analyze the errors—but it may alter its processing if there was an error. If you do not provide an error analysis routine, your program can issue SHOWCB or TESTCB to analyze an error when it gets control back following the CHECK.

Example 2: check return codes after a synchronous request

With synchronous processing, you should test register 15 after the request because the request may not have been accepted (register 15 contains 4) or because an error might have occurred (8 or 12):

```
         GET   RPL=REQPARMS
         LTR   15,15            Was request completed successfully?
         BNZ      REJFAIL       If branch is not taken, was request     x
                                accepted and completed successfully?
         ...
REJFAIL  ...
```

Example 3: overlap processing

In this example, the CHECK macro is used to wait for completion of a request before continuing to other processing. Access is asynchronous.

```
BLOCK    ACB
LIST     RPL   ACB=BLOCK,      Asynchronous access.                    x
               AREA=WORK,                                              x
               AREALEN=50,                                             x
               OPTCD=ASY
         ...
```

```
LOOP     GET   RPL=LIST                                                x
         LTR   15,15                                                   x
         BNZ   NOTACCEP
```

Do other processing:

```
         CHECK RPL=LIST     Suspends your processing to wait for com-   x
                            pletion of GET if necessary and to cause    x
                            VSAM to show return codes.
         LTR   15,15
         BNZ   ERROR
```

Process the record:

```
         B     LOOP
NOTACCEP ...                Request was not accepted.
```

```
ERROR    ...                Request failed.
         ...
WORK     DS    CL50         Work area.
```

After issuing the request, make sure that VSAM accepted it before you go on to other processing. When you have done as much other processing as you can, issue the CHECK macro. VSAM does not give you back control until the request is complete.

If you do not want to issue CHECK until you know the request is complete, use the ECB parameter of the RPL macro or the IO=COMPLETE parameter of the TESTCB macro. After you issue the CHECK, VSAM immediately returns a code and takes an exit, if necessary. See “RPL—Generate a request parameter list at assembly time” on page 70 and “GENCB—Generate a request parameter list at execution time” on page 40 for information on the ECB parameter.

Example 4: suspend a request for many records

In this example, a CHECK macro is issued for the first request parameter list in a chain of parameter lists. If an error occurred for one of the request parameter lists in the chain and you have supplied error analysis routines, VSAM takes a LERAD or SYNAD exit before it returns control to your program after the CHECK. For SYNAD exit routine, the CCHHR can be in CCCCcccHR format.

```
FIRST    RPL   ACB=BLOCK,                                              x
               AREA=AREA1,                                             x
               AREALEN=50,                                             x
               NXTRPL=SECOND,                                          x
               OPTCD=ASY
```

```
SECOND   RPL   ACB=BLOCK,                                              x
               AREA=AREA2,                                             x
               AREALEN=50,                                             x
               NXTRPL=THIRD,                                           x
               OPTCD=ASY

THIRD    RPL   ACB=BLOCK,    Last list does not indicate a next list.   x
               AREA=AREA3,                                             x
               AREALEN=50,                                             x
               OPTCD=ASY
         ...
LOOP     GET   RPL=FIRST     Request gives address of first request     x
                             parameter list.
         LTR   15,15
         BNZ   NOTACCEP
```

Do other processing:

```
         CHECK RPL=FIRST
         LTR   15,15
         BNZ   ERROR
```

Process the three records retrieved by the GET:

```
         B     LOOP
NOTACCEP ...                 Request wasn't accepted.

ERROR    ...                 Display feedback field (FIELDS=FDBK) of    x
                             each request list to determine which       x
                             one had an error.

AREA1    DS    CL50          A single GET request causes VSAM to put    x
                             a record in AREA1, AREA2, and AREA3.
AREA2    DS    CL50

AREA3    DS    CL50
```

After the CHECK, register 15 is set to indicate the status of the request. A code of 0 indicates that no error was associated with any of the request parameter lists. Any other code indicates that an error occurred for one of the request parameter lists. You should issue a SHOWCB macro for each request parameter list in the chain to find out which one had an error. VSAM does not process any of the request parameter lists beyond the one with an error.

CLOSE—Disconnect program and data

Use the CLOSE macro to disconnect the program and data.

If you use a "reserved" relative generation number character as the first character of a member name, the stow will not occur, you must issue your own stow.

The format of the CLOSE macro is:

[*label*]	CLOSE	(*address*[,[(*options*)][,...]]) [,MODE={24|31}] [,TYPE=T]

label
specifies 1 to 8 characters that provide a symbolic address for the CLOSE macro.

address
specifies the address of the access method control block or DCB for each data set to be closed. You may specify the address in register notation (using a register from 2 through 12—in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant. If you specify only one address with a register, you must enclose the expression identifying the register in two sets of parentheses: for example, CLOSE ((2)).

options
specifies options parameters for use only in closing non-VSAM data sets. If any options are specified with the address of an access method control block, VSAM ignores them.

Requirement: Because the CLOSE parameters are positional, include a comma for options (even if you do not specify options) before a subsequent parameter.

MODE={24|31}
specifies the format of the CLOSE parameter list to be built.

24
specifies a standard form (24-bit) parameter list address be built. This parameter list must reside below 16 megabytes and contain the address of ACBs residing below 16 megabytes. The caller, however, may be above 16 megabytes. This is the default parameter list format.

31
specifies a long form (31-bit) parameter list address be built. This list can reside above or below 16 megabytes. This value **must** be coded if the parameter list resides above 16 megabytes or contains the address of an VSAM/VTAM ACB residing above 16 megabytes.

TYPE=T
specifies VSAM is to complete outstanding I/O operations and update the catalog, but not disconnect the program from the data.

You can issue a temporary CLOSE macro to cause VSAM to complete outstanding I/O operations, put back into the catalog the updated information brought into virtual storage when the data set was opened, and write records in the SMF data set if you are using SMF. A temporary CLOSE does not disconnect the program from the data set, so your program can continue to process the data set without issuing an OPEN macro again.

You must close and reopen a newly allocated VSAM data set before you can issue noncreate requests. A temporary close is not adequate for this purpose.

The TYPE=T option does not release DASD space.

Requirement: If you are sharing subtasks or if you have issued an asynchronous request for access to a data set, you must issue a CHECK or an ENDREQ on all RPLs before you issue a CLOSE or CLOSE TYPE=T. Otherwise, concurrent data set I/O activity will cause unpredictable results during a close.

Example: CLOSE macro

This example shows how to close an ACB with a parameter list that may reside above 16 megabytes.

```
BLOCK1  ACB    .
               .
               RMODE31=ALL     VSAM control blocks and I/O buffers      x
               .               may be above 16 megabytes.
               .
        OPEN   BLOCK1,         OPEN/CLOSE parameter list may reside     x
               MODE=31         above 16 megabytes.

        CLOSE  BLOCK1,                                                  x
               MODE=31,                                                 x
               TYPE=T
```

The CLOSE parameters are:

- MODE=31 is required if the OPEN/CLOSE parameter list resides above 16 megabytes or if the ACB resides above 16 megabytes.
- TYPE indicates a temporary CLOSE. This causes VSAM to complete outstanding I/O operations, put back into the catalog the updated information that was brought into virtual storage when the data set was opened, and write records in the SMF data set if you are using SMF.

DLVRP—Delete VSAM resource pool

The DLVRP macro has a standard form and an execute form. The standard form builds a parameter list and passes control to VSAM to delete the resource pool.

The format of the DLVRP macro is:

[*label*]	DLVRP	TYPE={LSR|GSR} [,MODE={24|31}] [,SHRPOOL={0|*abs expression*}]

label
specifies 1 to 8 characters that provide a symbolic address for the DLVRP macro.

TYPE={LSR|GSR}
specifies the type of resource pool to be deleted: local (LSR) or global (GSR). When deleting an LSR pool, the number specified on the SHRPOOL parameter indicates which LSR pool is deleted. If both a data resource pool and an index resource pool have the same SHRPOOL number, both are deleted. The program that issues DLVRP TYPE=GSR must be in supervisor state with protection key 0 to 7.

MODE={24|31}
specifies the format of the DLVRP parameter list to be generated.

24
specifies that a standard form (24-bit) parameter list address be built. This parameter list must reside below 16 megabytes and contain the address of ACBs residing below 16 megabytes. The caller, however, may be above 16 megabytes. This is the default parameter list format.

31
specifies that a long form (31-bit) parameter list address be built. This list can reside above or below 16 megabytes. This parameter value **must** be coded if the parameter list resides above 16 megabytes or contains the address of a VSAM/VTAM ACB residing above 16 megabytes.

SHRPOOL={0|*abs expression*}
specifies the identification number of the shared resource pool to be deleted. Valid only when TYPE=LSR is also specified. The DLVRP parameter list may reside above or below 16 megabytes.

0
specifies the shared pool with the identification of 0. This is the default LSR pool identification number.

abs expression
specifies the shared pool with the identification of *abs expression* where *abs expression* is a number from 0 to 255.

Example: DLVRP macro

This example shows how an LSR pool with a parameter list that may reside above 16 megabytes and identification number other than 0 is deleted.

```
DELPOOL DLVRP  TYPE=LSR,                                              x
               MODE=31,                                               x
               SHRPOOL=1
```

The DLVRP parameters are:

- TYPE specifies that an LSR pool be deleted.
- MODE=31 specifies the parameter list may reside above or below 16 megabytes.
- SHRPOOL=1 specifies that both the data resource pool and the index resource pool (if any), with the identification number of 1, are to be deleted.

DLVRP—Execute form

The format of the execute form of DLVRP is:

[*label*]	DLVRP	MF=(E,*address*) [,SHRPOOL=*abs expression*] [,MODE={24|31}] ,TYPE={LSR|GSR}

If MODE=31 in the BLDVRP macro, then MODE=31 is required in the DLVRP macro.

There is no list form for DLVRP, because DLVRP works with BLDVRP: DLVRP uses the parameter list associated with BLDVRP. The address is the address of the parameter list built by a list form of BLDVRP. If you use register notation, use register 1 to pass the address of the parameter list to VSAM.

ENDREQ—Terminate a request

Use the ENDREQ macro to end a request, such as releasing exclusive control of a control interval containing a record.

The format of the ENDREQ macro is:

[*label*]	ENDREQ	RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the ENDREQ macro.

RPL=*address*
specifies the address of the request parameter list that defines the request. Specify the address either in register notation (using a register from 1 through 12, enclosed in parentheses) or as an RX-type address.

Requirement: The ENDREQ macro must not be issued when records are being loaded into a VSAM data set (load mode). ENDREQs issued while in load mode are not processed. ENDREQ will wait for the target RPL to post and, for that reason, it should not be issued in an attempt to terminate a hung request.

Example: release positioning for another request

In this example, the ENDREQ macro is used to cause VSAM to release exclusive control of a control interval containing a record. There are two request parameter lists, both of which require VSAM to be able to remember its position until VSAM is explicitly requested to forget its position.

```
BLOCK   ACB    MACRF=(SEQ,                                              x
               DIR),STRNO=2
SEQ     RPL    ACB=BLOCK,          VSAM must remember its position.     x
               OPTCD=SEQ
DIRUPD  RPL    ACB=BLOCK,          VSAM must remember its position and maintain  x
               OPTCD=(DIR,UPD)     exclusive control until explicitly requested  x
        .                          to forget it by PUT or ENDREQ.
        .
LOOP    GET    RPL=SEQ             VSAM now remembers its position for   x
                                   this request only while it is
        LTR    15,15               processing the request.
        BNZ    ERROR
        GET    RPL=DIRUPD          VSAM can remember its position for this  x
                                   request.
        LTR    15,15               The control interval will be placed in   x
                                   exclusive control until either
        BNZ    ERROR               ENDREQ or PUT UPD is issued.
```

Decide whether to update the record:

```
        B      FORGET              No; do not update the record.
        PUT    RPL=DIRUPD          Yes; update the record, causing VSAM     x
```

```
                                    to forget its position for DIRUP.
        LTR    15,15
        BNZ    ERROR
        B      LOOP
FORGET  ENDREQ RPL=DIRUPD          Cause VSAM to forget its position for DIRUPD.
        LTR    15,15               Release exclusive control.
        BNZ    ERROR
        B      LOOP
ERROR   xxx                        Request wasn't accepted or failed.
```

The use of ENDREQ shown here causes VSAM to release exclusive control of the control interval for a record. When PUT is issued after a DIRUPD GET request, ENDREQ need not be issued, because PUT causes VSAM to release exclusive control (the next DIRUPD GET does not depend on VSAM's remembering its position). Another result of ENDREQ is that current buffers are written if they have been modified.

To cause VSAM to give up its position associated with a chain of request parameter lists, specify the first request parameter list in the chain in your ENDREQ macro.

ENDREQ can also be used to cancel an asynchronous request, rather than suspending processing with CHECK.

Because VSAM remembers its position after a direct GET with OPTCD=UPD, LOC or (NUP, NSP), if no PUT or ENDREQ follows, you can switch to sequential access and use the positioning for a GET.

Requirement: If you are sharing subtasks or if you have issued an asynchronous request for access to a data set, you must issue a CHECK or an ENDREQ on all RPLs before you issue a CLOSE or CLOSE TYPE=T. Otherwise, concurrent data set I/O activity causes unpredictable results during a close.

ERASE—Delete a record

Use the ERASE macro to delete VSAM records. With ERASE processing of key-sequenced data sets or variable-length RRDS, VSAM attempts to make the control interval available to the control area when the last record in the control interval is erased. Thus, key-sequenced data set control intervals can be reused for new records whose keys fall anywhere within the control area's range of keys. The high key control interval of a control area is never reclaimed.

Variable-length RRDS control intervals can be reused for new records. The new variable-length RRDS record is inserted where the old record was, and the relative record number of the deleted record is reused for the new record.

ERASE is not supported for z/SO Unix files. You receive an error if you specify ERASE against a z/OS UNIX file.

To do an erase you must first do a GET UPD for the record.

The format of the ERASE macro is:

[*label*]	ERASE	RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the ERASE macro.

RPL=*address*
specifies the address of a request parameter list that defines the request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Example 1: keyed-direct deletion (KSDS, RRDS)

In this example, GET and ERASE macros are used to retrieve and delete records. Not every retrieved record is deleted. The search argument is a full key (5 bytes), compared equal.

```
DELETE   ACB   MACRF=(KEY,DIR,                                    x
               OUT)
```

```
LIST     RPL   ACB=DELETE,                                          x
               AREA=WORK,                                           x
               AREALEN=50,                                          x
               ARG=KEYFIELD,                                        x
               OPTCD=(KEY,DIR,                                      x
         .     SYN,UPD,             UPD indicates deletion.         x
         .     MVE,FKS,                                             x
         .     KEQ)
         .
LOOP     MVC   KEYFIELD,source      Search argument for retrieval,  x
                                    from table or transaction record.
         GET   RPL=LIST
         LTR   15,15
         BNZ   ERROR
```

Decide whether to delete the record:

```
         BE    LOOP                 No; retrieve the next record.
        ERASE  RPL=LIST             Yes; delete the record.
```

```
         LTR     15,15
         BNZ     ERROR
         B       LOOP
ERROR    ...                        Request not accepted, or failed.
WORK     DS      CL50               Examine the data record here.
KEYFIELD DS      CL5                Search argument.
```

When you retrieve a record for deletion (OPTCD=UPD, same as retrieval for update), VSAM is positioned at the record retrieved, in anticipation of a succeeding ERASE (or PUT) request for that record. However, you are not required to issue such a request. Another GET request nullifies any previous positioning for deletion or update.

Keyed-sequential retrieval for deletion varies from direct in that it does not use a search argument (except for possible use of the POINT macro). Skip-sequential retrieval for deletion (OPTCD=(SKP,UPD)) has the same effect as direct, but it is faster or slower depending on the number of control intervals separating the records being retrieved.

Example 2: addressed-sequential deletion (ESDS, KSDS)

In this example, the ERASE macro is used to delete records from a key-sequenced data set. Not every record retrieved for deletion is deleted. The POINT macro is used to skip records.

```
DELETE   ACB   MACRF=(ADR,SEQ,                                      x
               OUT)

REQUEST  RPL   ACB=DELETE,                                          x
               AREA=WORK,                                           x
               AREALEN=100,                                         x
               ARG=ADDR,                                            x
               OPTCD=(ADR,SEQ,                                      x
               ASY,UPD,MVE)         UPD indicates deletion.
               .
LOOP           ...                  Decide whether you need to skip    x
                                    to another position (forward or    x
                                    backward).

         B     RETRIEVE             No; bypass the POINT.

         MVC   ADDR,source          Yes; move search argument for      x
                                    POINT into search-argument field.

         POINT RPL=REQUEST          Position VSAM to the record to     x
                                    be retrieved next.
         LTR   15,15
         BNZ   ERROR
         CHECK RPL=REQUEST
```

```
         LTR   15,15
         BNZ   ERROR
RETRIEVE GET   RPL=REQUEST
         LTR   15,15
         BNZ   ERROR
```

```
         CHECK RPL=REQUEST
         LTR   15,15
         BNZ   ERROR
```

Decide whether to delete the record.

```
         BE    LOOP                No; skip ERASE and CHECK.
         ERASE RPL=REQUEST         Yes; delete the record.
         LTR   15,15
         BNZ   ERROR
         CHECK RPL=REQUEST
         LTR   15,15
         BNZ   ERROR
         B     LOOP
ERROR    ...                        Request not accepted, or failed.
         .
ADDR     DS    F                    RBA search argument for POINT.
WORK     DS    CL100                Work area.
```

Addressed deletion is allowed only for a key-sequenced data set. The records of an entry-sequenced data set are fixed. When records are deleted from a key-sequenced data set using addressed deletion, the index is not updated.

EXLST—Generate an exit list at assembly time

Use the EXLST macro to generate an exit list at assembly time. Values for EXLST macro subparameters can be specified as absolute numeric expressions, character strings, codes, and expressions that generate valid relocatable A-type address constants.

See: *z/OS DFSMS Using Data Sets* for the factors that determine the addressing mode and the parameter list residency mode set when the exit routine gets control.

The format of the EXLST macro is:

[*label*]	EXLST	[AM= VSAM] [,EODAD=(*address*[,A|N][,L])] [,JRNAD=(*address*[,A|N][,L])] [,LERAD=(*address*[,A|N][,L])] [,SYNAD=(*address*[,A|N][,L])] [,UPAD=(*address*[,A|N][,L])] [RLSWAIT=(*address*[,A|N][,L])]

label
specifies 1 to 8 characters that provide a symbolic address for the established exit list.

AM=VSAM
specifies that the access method using the control block is VSAM.

EODAD=(*address*[,A|N][,L])
JRNAD=(*address*[,A|N][,L])
LERAD=(*address*[,A|N][,L])
SYNAD=(*address*[,A|N][,L])
UPAD=(*address*[,A|N][,L])

RLSWAIT=(*address*[,A|N][,L])
specify that you are supplying a routine for the exit specified.

For more information about user exit routines, see *z/OS DFSMS Using Data Sets*.

The exits and values that can be specified for these routines are:

EODAD
specifies that an exit is provided for special processing when the end of a data set is reached by sequential access.

JRNAD
specifies that an exit is provided for journalizing transactions as you process data records. For RLS, JRNAD is not supported and you receive an error if you open the ACB. This parameter has no effect for z/OS UNIX files.

LERAD
specifies that an exit is provided for analyzing logical errors.

SYNAD
specifies that an exit is provided for analyzing physical errors.

UPAD
specifies that an exit is provided for user processing during a VSAM request. The GENCB, MODCB, SHOWCB, and TESTCB macros do not support the UPAD user exit routine. For RLS, UPAD is ignored and the RLSWAIT exit is used instead. This parameter has no effect for z/OS UNIX files.

RLSWAIT
For RLS, this exit is used instead of UPAD. If you specify a UPAD exit for RLS. it is ignored. The RLSWAIT exit is specified on an ACB basis and is entered in 31 bit mode. When the exit is to be used for a record management request the RPL must specify OPTCD=(SYN,WAITX). The RLSWAIT exit is entered after an asynchronous execution unit is scheduled to process the request. The exit is intended for those applications which issue VSAM RLS requests and can not tolerate VSAM suspending the execution unit which issued the record management request.

address
specifies the address of a user-supplied exit routine or an I/O prevention identifier. The address must immediately follow the equal sign.

A|N
specifies that the exit routine is active (A) or not active (N). VSAM does not enter a routine whose exit is marked not active.

L
specifies that the address is an 8-byte field containing the name of an exit routine in a partitioned data set identified by a JOBLIB or STEPLIB DD statement or in SYS1.LINKLIB. VSAM loads the exit routine for exit processing. If **L** is omitted, the address gives the entry point of the exit routine in virtual storage, and the exit routine is entered in the addressing mode of the VSAM caller.

Requirement: The EXLST macro generates an exit list with each entry 5 bytes in length. You must consider the proper alignment of any subsequent data.

Example: EXLST macro

An EXLST macro is used to identify exit routines provided for analyzing logical and physical errors. The label of the EXLST macro (EXITS) is used in an ACB or GENCB macro that generates an access method control block to associate the exit list with an access method control block. The exit list generated by this example is built when the program is assembled.

```
EXITS    EXLST EODAD=(ENDUP,N),        EXITS gives symbolic address     x
               LERAD=LOGICAL,          of the exit list.                x
               SYNAD=(ROUTNAME,L)
ENDUP                                  EODAD routine.
LOGICAL                                LERAD routine.
ROUTNAME DC    C'PHYSICAL'             Pad shorter names with           x
                                       blanks:C'SYN     ' or CL8'SYN'.
```

The EXLST macro's parameters are:

- EODAD specifies that the end-of-data routine is located at ENDUP and is not active.
- LERAD specifies that the logical error routine is located at LOGICAL and is active.
- SYNAD specifies that the physical error routine's name is located at ROUTNAME.

GENCB—Generate an access method control block at execution time

The format of the GENCB macro used to generate an access method control block is:

[*label*]	GENCB	BLK=ACB [,AM=VSAM] [,BSTRNO=*abs expression*] [,BUFND=*abs expression*] [,BUFNI=*abs expression*] [,BUFSP=*abs expression*] [,COPIES=*abs expression*] [,DDNAME=*character string*] [,EXLST= *address*] [,LENGTH=*abs expression*] [,LOC=BELOW|ANY] [,MACRF=([ADR][,CNV][,KEY] [,CFX|NFX] [,DDN|DSN] [,DFR|NDF] [,DIR][,SEQ][,SKP] [,ICI|NCI] [,IN][,OUT] [,LEW|NLW] [,NIS|SIS] [,NRM|AIX] [,NRS|RST] [,NSR|LSR|GSR|RLS] [,NUB|UBF])] [,DB={YES|NO}] [,MAREA=*address*] [,MLEN=*abs expression*] [,PASSWD=*address*] [,RMODE31={ALL|BUFF|CB|NONE}] [,SHRPOOL={0|*abs expression*}] [,STRNO=*abs expression*] [,RLSREAD={NRI|CR|NORD}] [,WAREA=*address*] [,ZHYPERWRITE={YES|NO}]

The subparameters of the GENCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. “Subparameters with GENCB, MODCB, SHOWCB, and TESTCB” on page 5, further defines these operand expressions.

label
specifies 1 to 8 characters that provide a symbolic address for the GENCB macro.

BLK=ACB
specifies that you are generating an access method control block.

AM=VSAM
specifies that the access method using this control block is VSAM.

BSTRNO=*abs expression*
specifies the number of strings initially allocated for access to the base cluster of a path. BSTRNO must be a number between 0 and 255. The default is STRNO. BSTRNO is ignored if the object being opened is not a path. If the number specified for BSTRNO is insufficient, VSAM dynamically extends the number of strings as needed for the access to the base cluster. BSTRNO can also influence performance. The VSAM control blocks for the set of strings specified by BSTRNO are allocated

on contiguous virtual storage, whereas this is not guaranteed for the strings allocated by dynamic extension.

For RLS, BSTRNO is ignored. This parameter has no effect for z/OS UNIX files.

BUFND=*abs expression*
specifies the number of I/O buffers VSAM uses for transmitting data between virtual and auxiliary storage. A buffer is the size of a control interval in the data component. BUFND must be a number between 0 and 32767. The minimum number you may specify is 1 plus the number specified for STRNO (if you omit STRNO, BUFND must be at least 2, because the default for STRNO is 1). The number can be supplied through the JCL DD AMP parameter and through the macro. The default is the minimum number required. A larger number for BUFND can improve the performance of sequential access.

For RLS, BUFND is ignored. This parameter has no effect for z/OS UNIX files.

BUFNI=*abs expression*
specifies the number of I/O buffers VSAM uses for transmitting index entries between virtual and auxiliary storage for keyed access. A buffer is the size of a control interval in the index. BUFNI must be a number between 0 and 32767. The minimum number is the number specified for STRNO (if you omit STRNO, BUFNI must be at least 1, because the default for STRNO is 1). You can supply the number through the JCL DD AMP parameter and through the macro. The default is the minimum number required. A larger number for BUFNI can improve the performance of keyed-direct retrieval.

For RLS, BUFNI is ignored. This parameter has no effect for z/OS UNIX files.

BUFSP=*abs expression*
specifies the maximum number of bytes of virtual storage used for the data and index I/O buffers. VSAM gets the storage in your program's address space. If you specify less than the amount of space specified in the BUFFERSPACE parameter of the DEFINE command when the data set was defined, VSAM overrides your BUFSP specification upward to the value specified in BUFFERSPACE. (BUFFERSPACE, by definition, is the least amount of virtual storage that is ever provided for I/O buffers.) You can supply BUFSP through the JCL DD AMP parameter and through the macro. If you do not specify BUFSP in either place, the amount of storage used for buffer allocation is the *largest* of:

- The amount specified in the catalog (BUFFERSPACE),
- The amount determined from BUFND and BUFNI, or
- The minimum storage required to process the data set with its specified processing options.

If BUFSP is specified and the amount is smaller than the minimum amount of storage required to process the data set, VSAM cannot open the data set.

A valid BUFSP amount takes precedence over the amount called for by BUFND and BUFNI. If the BUFSP amount is greater than the amount called for by BUFND and BUFNI, the extra space is allocated as follows:

- When MACRF indicates direct access only, additional index buffers are allocated.
- When MACRF indicates sequential access, one additional index buffer and as many data buffers as possible are allocated.

If the BUFSP amount is less than the amount called for by BUFND and BUFNI, the number of data and index buffers is decreased as follows:

- When MACRF indicates direct access only, the number of data buffers is decreased to not less than the minimum number. Then, if required, the number of index buffers is decreased until the amount called for by BUFND and BUFNI complies with the BUFSP amount.
- When MACRF indicates sequential access, the number of index buffers is decreased to not less than 1 more than the minimum number. Then, if required, the number of data buffers is decreased to not less than the minimum number. If still required, 1 more is subtracted from the number of index buffers.
- Neither the number of data buffers nor the number of index buffers is decreased to less than the minimum number.

If the index does not exist or is not being opened, only BUFND, and not BUFNI, enters into these calculations.

For RLS, BUFSP is ignored. This parameter has no effect for z/OS UNIX files.

COPIES=*abs expression*
specifies the number of copies of the access method control block VSAM generates. All the copies are identical. Use MODCB to tailor the individual copies for particular data sets and processing. MODCB is described in "MODCB—Modify an access method control block" on page 54.

DDNAME=*character string*
specifies 1 to 8 characters that identify the data set you want to process by specifying the JCL DD statement for the data set. You may omit DDNAME and provide it through the MODCB macro before opening the data set. MODCB is described in "MODCB—Modify an access method control block" on page 54.

EXLST=*address*
specifies the address of a list of addresses of exit routines you are providing. The list is established by the EXLST or GENCB macro. If you use the EXLST macro, you can specify its label here as the address of the exit list. If you use GENCB, you can specify the address returned by GENCB in register 1. Omitting this parameter indicates that you have no exit routines. VSAM user exit routines are described in *z/OS DFSMS Using Data Sets*.

LENGTH=*abs expression*
specifies the length, in bytes, of the area, if any, you are supplying for VSAM to generate the access method control blocks. (See the WAREA parameter.) The LENGTH value cannot exceed 65535 (X'FFFF').

LOC={BELOW|ANY}

BELOW
specifies that VSAM is to construct an ACB in an area of virtual storage below 16 megabytes at execution time. This is the default.

ANY
specifies that VSAM is to construct an ACB in an area of virtual storage above 16 megabytes, if possible, at execution time.

The LOC parameter is different from other GENCB parameters. If you code it on the list form, the execute form always overrides it. If you want LOC=ANY when using the list and execute forms, you must code it on the execute form. For more information, refer to "GENCB—List form" on page 46 and "GENCB—Execute form" on page 46.

MACRF=([ADR][,CNV][,KEY]
[,CFX|NFX]
[,DDN|DSN]
[,DFR|NDF]
[,DIR][,SEQ][,SKP]
[,ICI|NCI]
[,IN][,OUT]
[,LEW|NLW]
[,NIS|SIS]
[,NRM|AIX]
[,NRS|RST]
[,NSR|LSR|GSR|RLS]
[,NUB|UBF])
specifies the kinds of processing you will do with the data set. The subparameters must be significant for the data set. For example, if you specify keyed access for an entry-sequenced data set, you cannot open the data set. You must specify all the types of access you are going to use, whether you use them concurrently or by switching from one to the other. The subparameters are shown in Table 2 on page 12. They are arranged in groups, and each group has a default value (shown by underlining). You may specify subparameters in any order. You may specify both ADR and KEY to process a key-sequenced data set. You may specify both DIR and SEQ; with keyed access, you may

specify SKP as well. If you specify OUT and want merely to retrieve some records and also update, delete, or insert others, you need not also specify IN.

DB={YES|NO}

YES
specifies that RLS processes the VSAM database (VSAMDB) as a database, honoring the DATABASE specification in DEFINE CLUSTER or DEFINE AIX, not as a regular KSDS. YES is the default.

NO
specifies that RLS processes the VSAM database (VSAMDB) as a regular KSDS, not as a database, ignoring the DATABASE specification in DEFINE CLUSTER or DEFINE AIX.

Note: Non-RLS VSAM always treats a VSAM database as a regular KSDS, ignoring DB=[YES|NO] and the DATABASE specification in DEFINE CLUSTER or DEFINE AIX.

MAREA=*address*
specifies the address of an optional OPEN/CLOSE or TYPE=T option (CLOSE macro) message area. See "OPEN/CLOSE message area for multiple reason or attention messages" on page 116.

MAREA is ignored for RLS processing.

MLEN=*abs expression*
specifies the length of an optional OPEN/CLOSE or TYPE=T option (CLOSE macro) message area.

MLEN is ignored for RLS processing.

PASSWD=*address*
specifies the address of a field that contains the highest-level password required for the types of access indicated by the MACRF parameter. The first byte of the field contains the length (in binary) of the password (maximum of 8 bytes). Zero indicates that no password is supplied. If the data set is password protected and you do not supply a required password in the access method control block, VSAM may give the console operator the opportunity to supply it when you open the data set. This parameter has no effect for z/OS UNIX files.

RLSREAD={NRI|CR|NORD}
RLSREAD (for RLS), specifies the read integrity options that apply to GET requests issued against this ACB. This parameter overrides the read integrity options specified in the RLS JCL parameter. Read integrity options can also be specified on the GET request, when they override the RLSREAD specification.

NRI
specifies no read integrity.

CR
specifies consistent read integrity.

NORD
species the read integrity option used is determined either by the RLS JCL specification or by options specified on the GET request.

For non-RLS, this parameter is ignored.

RMODE31={ALL|BUFF|CB|NONE}
specifies where VSAM OPEN is to obtain virtual storage (above or below 16 megabytes) for control blocks and I/O buffers.

The values specified by the RMODE31 parameter only have an effect on VSAM at the setting just before an OPEN is issued. At all other times, changing these values has no effect on the residency of the control blocks and I/O buffers.

The virtual storage location of the ACB is independent of the RMODE31 parameter. An ACB may reside either above or below 16 megabytes.

RMODE31 is ignored for RLS processing.

ALL
specifies both VSAM control blocks and I/O buffers are obtained above 16 megabytes.

BUFF
specifies only VSAM I/O buffers are obtained above 16 megabytes.

CB
specifies only VSAM control blocks are obtained above 16 megabytes.

NONE
specifies both VSAM control blocks and I/O buffers are obtained below 16 megabytes. This is the default.

SHRPOOL={*abs expression*|0}
specifies the identification number of the resource pool used for LSR processing. SHRPOOL must be a number between 0 and 255. The default is SHRPOOL=0. For RLS, SHRPOOL is ignored. This parameter has no effect for z/OS UNIX files.

STRNO=*abs expression*
specifies the number of requests requiring concurrent data set positioning VSAM is prepared to handle. A request is defined by a given request parameter list or chain of request parameter lists. STRNO must be a number between 1 and 255. See "RPL—Generate a request parameter list at assembly time" on page 70 and "GENCB—Generate a request parameter list at execution time" on page 40 for information on request parameter lists. For RLS, STRNO is ignored and strings are dynamically acquired up to a limit of 1024. STRNO > 1 is not supported for z/OS UNIX files and, if specified with a value greater than 1, results in an open failure.

WAREA=*address*
specifies the address of an area in which to generate the access method control blocks.

The area must begin on a fullword boundary.

This parameter is paired with the LENGTH parameter. You must supply the LENGTH parameter if you specify an area address.

If you do not specify an area in which the access method control block is to be generated, VSAM obtains virtual storage space for the area (as specified by the LOC=keyword). Subpool 0 will be requested under the user's key and state. Users executing in key 0 and supervisor state will actually be assigned subpool 252. VSAM returns the address of the area containing the control blocks in register 1 and the length of the area in register 0. You can determine the length of each control block by dividing the length of the area by the number of copies. The address of each control block can then be calculated by this offset from the address in register 1. You can find the length of an access method control block with the SHOWCB macro.

If you are generating control blocks by issuing several GENCBs, specifying an area (WAREA and LENGTH parameters) for them allows you to address all of them with one base register and to avoid repetitive requests for virtual storage.

ZHYPERWRITE={YES|NO}
specifies YES to enable zHyperWrite support for the data set.

Example: GENCB macro (generate an access method control block)

In this example, a GENCB macro is used to identify a data set to open and to specify the types of processing to perform. This example specifies that the space for the control block be obtained above 16 megabytes. The access method control block generated by this example is built when the program is executed.

```
GENCB  GENCB   BLK=ACB,AM=VSAM,     One copy generated; VSAM gets the   x
               BUFND=4,BUFNI=3,     storage for it, because the         x
               BUFSP=19456,         WAREA LENGTH parameters have        x
               DDNAME=DATASETS,     been omitted.                       x
               EXLST=EXITS,                                             x
               LOC=ANY,                                                 x
               MACRF=(KEY,DIR,                                          x
               SEQ,OUT),                                                x
               RMODE31=ALL,                                             x
```

```
               STRNO=2


         ST    1,ACBADDR           Save the address of the access      x
                                   method control block.
ACBADDR  DS    A                   The address of the access method    x
                                   control block is saved in ACBADDR.
```

The GENCB macro's parameters are:

- BUFND specifies four I/O buffers for data. BUFNI specifies three I/O buffers for index entries. BUFSP specifies 19456 bytes of buffer space, enough space to accommodate control intervals of data that are 4096 bytes and of index entries that are 1024 bytes.
- DDNAME specifies that this access method control block is associated with a DD statement named DATASETS.
- EXLST specifies that the exit list associated with this access method control block is named EXITS.
- LOC specifies that VSAM obtain virtual storage for the ACB from an area that may be above 16 megabytes.
- MACRF specifies keyed direct and keyed sequential processing for both insertion and update.
- RMODE31 specifies that VSAM obtain storage for the VSAM control blocks and I/O buffers in an area above 16 megabytes when the ACB is opened.
- STRNO specifies that two requests will require concurrent positioning.

Example: GENCB macro (generate an access method control block)

The access method control block (ACB) generated by this example is built when the program is executed. In this example, the user provides the storage to contain the ACB. Because the generate form of the macro is used, the GENCB parameter list is built in a remote area and passed to VSAM for action.

```
         LA      10,LEN1             Get length of the GENCB parameter
                                     list returned by the GENCB macro.
         GETMAIN R,LV=(10)           Get storage for the area in which
                                     the GENCB parameter list is to
                                     be built.
         LR      2,1                 Save addr of GENCB parameter-list
                                     area.
         LA      10,ACBLNGTH         Get length of the ACB.

         GETMAIN R,LV=(10)           Get storage for the area in which
                                     the ACB is to be built.
         LR      3,1                 Save address of ACB area.

GENCB1   GENCB   BLK=ACB,AM=VSAM,    One copy generated; VSAM builds    x
                 BUFND=4,BUFNI=3,    the ACB in the storage provided    x
                 BUFSP=19456,        at the location pointed to by      x
                 DDNAME=DATASETS,    WAREA.                             x
                 LENGTH=ACBLNGTH,                                       x
                 MACRF=(KEY,DIR,                                        x
                 SEQ,OUT),                                              x
                 RMODE31=ALL,                                           x
                 WAREA=(3),                                             x

                 MF=(G,(2),LEN1)
                 .
                 .
                 .
ANYNAME  DSECT   KEEP ACB model out of CSECT
ACBSTART ACB     AM=VSAM
ACBEND   DS      0F
ACBLNGTH EQU     ACBEND-ACBSTART
```

The GENCB macro's parameters are:

- BUFND specifies four I/O buffers for data. BUFNI specifies three I/O buffers for index entries. BUFSP specifies 19456 bytes of buffer space, enough space to accommodate control intervals of data that are 4096 bytes and of index entries that are 1024 bytes.

- DDNAME specifies that this access method control block is associated with a DD statement named DATASETS.
- LENGTH specifies that the length of the storage you provide for the ACB is the value of ACBLNGTH.
- MACRF specifies keyed direct and keyed sequential processing for both insertion and update.
- RMODE31 specifies that VSAM obtain storage for the VSAM control blocks and I/O buffers in an area above 16 megabytes when the ACB is opened.
- WAREA specifies that the address of the storage you provide for the ACB is held in register 3.
- MF specifies that the GENCB parameter list is to be built in the location specified by register 2. Also, the expansion of the GENCB macro will equate LEN1 to the length of the GENCB parameter list.

GENCB—Generate an exit list at execution time

The format of the GENCB macro used to generate an exit list is:

[*label*]	GENCB	BLK=EXLST [,AM=VSAM] [,COPIES=*abs expression*] [,EODAD=(*address*[,A|N][,L])] [,JRNAD=(*address*[,A|N][,L])] [,LENGTH=*abs expression*] [,LERAD=(*address*[,A|N][,L])] [,LOC=BELOW|ANY] [,SYNAD=(*address*[,A|N][,L])] [,RLSWAIT=(*address*[,A|N][,L])] [,WAREA=*address*]

The subparameters of the GENCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Subparameters with GENCB, MODCB, SHOWCB, and TESTCB" on page 5, further defines these operand expressions.

See: *z/OS DFSMS Using Data Sets* for the factors that determine the addressing mode and the parameter list residency mode set when the exit routine gets control.

label
specifies 1 to 8 characters that provide a symbolic address for the GENCB macro.

BLK=EXLST
specifies that you are generating an exit list.

AM=VSAM
specifies that the access method using this control block is VSAM.

[,EODAD=(*address*[,A|N][,L])]
[,JRNAD=(*address*[,A|N][,L])]
[,LERAD=(*address*[,A|N][,L])]
[,SYNAD=(*address*[,A|N][,L])]
[,RLSWAIT=(*address*[,A|N][,L])]
specifies that you are supplying a routine for the exit named.

For more information about user exit routines, see *z/OS DFSMS Using Data Sets*.

If none of these user exit routines is specified, VSAM generates an exit list with inactive entries for all the exits. The exits and values that can be specified for them are:

COPIES=*abs expression*
specifies the number of copies of the exit list you want generated. GENCB generates as many copies as you specify (default is 1) when your program is executed. All copies are the same. You

can use MODCB to change some or all of the addresses in a list. MODCB is described in "MODCB—Modify an access method control block" on page 54.

EODAD
specifies that an exit is provided for special processing when the end of a data set is reached by sequential access.

JRNAD
specifies that an exit is provided for journaling as you process data records. For RLS, JRNAD is not supported and you receive an error if you open the ACB. This parameter has no effect for z/OS UNIX files.

LERAD
specifies that an exit is provided for analyzing logical errors.

SYNAD
specifies that an exit is provided for analyzing physical errors.

RLSWAIT
specifies that an exit is provided for wait processing. For RLS the UPAD exit is ignored if it is specified, and the RLSWAIT exit is used to perform a similar function.

address
specifies the address of a user-supplied exit routine. The address must immediately follow the equal sign.

A|N
specifies that the exit routine is active (A) or not active (N). VSAM does not enter a routine whose exit is marked not active.

L
specifies the address is an 8-byte field containing the name of an exit routine in a partitioned data set identified by a JOBLIB or STEPLIB DD statement or in SYS1.LINKLIB. VSAM is to load the exit routine for exit processing. If **L** is omitted, the address gives the entry point of the exit routine in virtual storage, and the exit routine is entered in the addressing mode of the VSAM caller.

L may precede or follow the **A** or **N** specification.

LENGTH=*abs expression*
specifies the length, in bytes, of the area, if any, that you are supplying for VSAM to generate the exit lists. (See the WAREA parameter.) The LENGTH value cannot exceed 65535 (X'FFFF').

LOC=BELOW|ANY

BELOW
specifies VSAM is to construct an exit list in an area below 16 megabytes at execution time.

ANY
specifies VSAM is to construct an exit list in an area above 16 megabytes, if possible, at execution time.

The LOC parameter is different from other GENCB parameters. If you code it on the list form, the execute form always overrides it. If you want LOC=ANY when using the list and execute forms, you must code it on the execute form. For more information, refer to "GENCB—List form" on page 46 and "GENCB—Execute form" on page 46.

WAREA=*address*
specifies the address of an area in which to generate the exit lists.

If you did not specify an area in which the exit list is to be generated, VSAM obtains virtual storage space for the area (as specified by the LOC=keyword). Subpool 0 will be requested under the user's key and state. Users executing in key 0 and supervisor state will actually be assigned subpool 252. VSAM returns the address of the area in which the exit lists is to be generated in register 1, and the length of the area in register 0. You can find the length of each exit list by dividing the length of the area by the number of copies. The address of each exit list can then be calculated by this offset from the address in register 1. You can find the length of an exit list with the SHOWCB macro, described under

"SHOWCB—Display fields of an exit list" on page 91.

If you are generating control blocks by issuing several GENCBs, specifying an area (WAREA and LENGTH) for them allows you to address all of them with one base register and to avoid repetitive requests for virtual storage.

Example: GENCB macro (generate an exit list)

In this example, a GENCB macro is used to generate an exit list when the program is executed.

```
EXITS    GENCB BLK=EXLST,                                              x
               EODAD=(EOD,N),                                          x
               LERAD=LOGICAL,                                          x
               SYNAD=(ERROR,                                           x
               A,L)

         LTR   15,15
         BNZ   ERROR
         ST    1,EXLSTADR         Address of the exit list is saved.
EOD      EQU   *                  EODAD routine.
LOGICAL  EQU   *                  LERAD routine.
ERROR    DC    C'PHYSICAL'        Name of the SYNAD module.
EXLSTADR DS    A                  Save area for exit-list address.
```

The GENCB macro's parameters are:

- BLK specifies an exit list is generated.
- EODAD specifies the end-of-data routine is located at EOD and is not active.
- LERAD specifies that the logical error routine is located at LOGICAL. Because neither A nor N is specified, the LERAD routine is marked active by default.
- SYNAD specifies that the physical error routine's name is located at ERROR.

Because no area is specified in which the exit list is to be generated, VSAM obtains virtual storage for the exit list and returns the address in register 1. Immediately after the GENCB macro, the address of the exit list, contained in register 1, is moved to EXLSTADR. EXLSTADR may be specified in a GENCB macro that generates an access method control block or in a MODCB, SHOWCB, or TESTCB macro that modifies, displays, or tests fields in an exit list.

GENCB—Generate a request parameter list at execution time

The format of the GENCB macro used to generate a request parameter list is:

[*label*]	GENCB	BLK=RPL [,ACB=*address*] [,AM=VSAM] [,AREA=*address*] [,AREALEN=*abs expression*] [,ARG=*address*] [,COPIES=*abs expression*] [,TIMEOUT=*number*] [,ECB=*address*] [,KEYLEN=*abs expression*] [,LENGTH=*abs expression*] [,LOC=BELOW|ANY] [,MSGAREA=*address*] [,MSGLEN=*abs expression*] [,NXTRPL=*address*] [,OPTCD=([ADR|CNV|KEY] [,DIR|SEQ|SKP] [,ARD|FRD|LRD] [,FWD|BWD] [,ASY|SYN] [,NSP|NUP|UPD] [,KEQ|KGE] [,FKS|GEN] [,LOC|MVE] [,NRI|CR] [,ARA31|ARA64] [,RBA|XRBA])] [,RECLEN=*abs expression*] [,DBARGLN=*abs expression*] [,TRANSID=*abs expression*] [,WAREA=*address*] [,ZHYPERWRITE={YES|NO}]

The subparameters of the GENCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. “Subparameters with GENCB, MODCB, SHOWCB, and TESTCB” on page 5, further defines these operand expressions.

The parameters of the GENCB macro to generate a request parameter list are optional sometimes, but required in others. It is not necessary to omit parameters that are not required for a request; they are ignored. Thus, if you switch from direct to sequential retrieval with a request parameter list, you do not have to zero out the address of the field containing the search argument (ARG=address).

label
specifies 1 to 8 characters that provide a symbolic address for the GENCB macro. For addressing lists generated by GENCB, see the COPIES parameter.

BLK=RPL
specifies you are generating a request parameter list.

ACB=*address*
specifies the address of the access method control block that identifies the data set to which access will be requested. If you omit this parameter, you must issue MODCB to specify the address of the access method control block before you issue a request. MODCB is described in “MODCB—Modify an access method control block” on page 54.

AM=VSAM
specifies that the access method using this control block is VSAM.

AREA=*address*
specifies the address of a work area to and from which VSAM moves a data record if you request it to do so (with the RPL parameter OPTCD=MVE). If you request that records be processed in the I/O buffer (OPTCD=LOC), VSAM puts into this work area the address of a data record within the I/O buffer.

AREALEN=*abs expression*
specifies the length, in bytes, of the work area whose address is specified by the AREA parameter. Its minimum for OPTCD=MVE is the size of a data record (or the largest data record, for a data set with records of variable length). For OPTCD=LOC, the area should be 4 bytes to contain the address of a data record within the I/O buffer.

ARG=*address*
specifies the address of a field containing the search argument for direct retrieval, skip-sequential retrieval, and positioning. For a fixed-length or variable-length RRDS, the ARG field must be 4 bytes long. For direct or skip-sequential processing, this field contains your search argument, a relative record number. For sequential processing (OPTCD=(KEY,SEQ)), the 4 bytes are required for VSAM to return the feedback RRN. For keyed access (OPTCD=KEY), the search argument is a full or generic key. For addressed access (OPTCD=ADR), the search argument is an RBA. If you specify a generic key (OPTCD=GEN), you must also specify in the KEYLEN parameter how many of the bytes of the full key you are using for the generic key.

For a VSAMDB base or AIX document in BSON data format, the search argument located at ARG=address must be a full search 'element' in the formal definition of BSON or JSON. See BSONspec.org and JSON.org for the formal definition of BSON and JSON data formats.

For BSON, the argument must be 'type, Key name, Key value length, Key value', such as '024E616D65000B0000004A6F686E20536D69746800' where type=02 (string), key name=4E616D6500 ('Name' in UTF-8), key value length=0B000000 (decimal 11), key value=4A6F686E20536D69746800 ('John Smith').

For JSON, the same search argument would be simply "John Smith", including the "s if it is a string type.

COPIES=*abs expression*
specifies the number of copies of the request parameter list to generate. GENCB generates as many copies as you specify (default is 1) when your program is executed.

The copies of a request parameter list can be used to:

- Chain lists together to gain access to many records with one request
- Define many requests to gain access to many parts of a data set concurrently.

All copies generated are identical; you must use MODCB to tailor them to specific requests. MODCB is described in “MODCB—Modify an access method control block” on page 54.

DBARGLN=*abs expression*
specifies the length, in bytes, of the search argument in the field that is specified in ARG=address for a VSAMDB database. It is a required parameter for POINT and GET DIR requests against VSAMDB JSON databases; DBARGLN is optional for BSON. DBARGLN is ignored for non-VSAMDB data sets.

ECB=*address*
specifies the address of an event control block (ECB) that you may supply. VSAM indicates in the ECB whether a request is complete or not (using standard completion codes, which are described in *z/OS MVS System Codes*). You can use the ECB to determine that an asynchronous request is complete before issuing a CHECK macro. This parameter is always optional.

KEYLEN=*abs expression*
specifies the length, in bytes, of the generic key (OPTCD=GEN) you are using for a search argument (given in the field addressed by the ARG parameter). This parameter is required with a search argument that is a generic key. The number can be 1 through 255. For full-key searches, VSAM knows the key length, which is taken from the catalog definition of the data set when you open the data set. This parameter has no effect for z/OS UNIX files.

LENGTH=*abs expression*
specifies the length, in bytes, of the area, if any, that you are supplying for VSAM to generate the request parameter lists. (See the WAREA parameter.) The LENGTH value cannot exceed 65535 (X'FFFF').

You can find out how long a request parameter list is with the SHOWCB macro, described in "SHOWCB—Display fields of a request parameter list" on page 92.

LOC=BELOW|ANY

BELOW
specifies that storage for the RPL be obtained from virtual storage below 16 megabytes.

ANY
specifies that storage be obtained from virtual storage above 16 megabytes if possible.

The LOC parameter is different from other GENCB parameters. If you code it on the list form, the execute form always overrides it. If you want LOC=ANY when using the list and execute forms, you must code it on the execute form. For more information, refer to "GENCB—List form" on page 46 and "GENCB—Execute form" on page 46.

MSGAREA=*address*
specifies the address of an area you are supplying for VSAM to send you a message if a physical error occurs. The format of a physical error message is given under "Reason code (physical errors)" on page 138 in the chapter Chapter 3, "VSAM macro return and reason codes," on page 107.

MSGLEN=*abs expression*
specifies the size, in bytes, of the message area indicated in the MSGAREA parameter. The size of a message is 128 bytes. If you provide less than 128 bytes, no message is returned to your program. This parameter is required when MSGAREA is coded.

NXTRPL=*address*
specifies the address of the next request parameter list in a chain. Omit this parameter from the macro that generates the only or last list in the chain. When you issue a request defined by a chain of request parameter lists, indicate in the request macro the address of the first parameter list in the chain. A single request macro can be defined by multiple request parameter lists. For example, a GET can cause VSAM to retrieve two or more records. This parameter has no effect for z/OS UNIX files, and if it is specified with a non-zero value, results in an error on a subsequent GET, PUT, or POINT.

OPTCD=([ADR|CNV|KEY]
[,DIR|SEQ|SKP]
[,ARD|FRD|LRD]
[,FWD|BWD]
[,ASY|SYN]
[,NSP|NUP|UPD]
[,KEQ|KGE]
[,FKS|GEN]
[,LOC|MVE])
[,CR|NRI]
[,ARA31|ARA64]
[,RBA|XRBA])
specifies the subparameters that govern the request defined by the request parameter list. Each group of subparameters has a default; subparameters are shown in Table 3 on page 72 with defaults underlined. Only one subparameter from each group is effective for a request. Some requests do not require an subparameter from all of the groups to be specified. The groups that are not required are ignored. Thus, you can use the same request parameter list for a combination of requests (GET, PUT, POINT, for example) without zeroing out the inapplicable subparameters each time you go from one request to another.

RECLEN=*abs expression*
specifies the length, in bytes, of a data record being stored. If the records you are storing are all the same length, you do not need to change RECLEN after you set it. This parameter is required for PUT

requests. For GET requests, VSAM puts the length of the record retrieved in this field in the request parameter list. It will be there if you update and store the record.

TIMEOUT=*number*
For RLS only, specifies the time in seconds that your program is willing to wait to obtain a lock on a VSAM record when a lock on the record is already held by another program.

A non-zero value for TIMEOUT (or if TIMEOUT is not specified) specifies the time (in seconds) this program waits for the other program(s) to release the lock.

A value of zero specifies TIMEOUT processing is *NOT* to be performed by VSAM for this request. That is, if the record lock required by the request is held by another program, the program waits until the other program releases the lock regardless of how long that might be.

TRANSID=*abs expression*
specifies a number that relates modified buffers in a buffer pool. Use in shared resource applications and a description are in *z/OS DFSMS Using Data Sets*. This parameter has no effect for z/OS UNIX files.

WAREA=*address*
specifies the address of an area in which the request parameter lists are generated.

If you did not specify an area in which the request parameter list is to be generated, VSAM obtains virtual storage space for the area (as specified by the LOC=keyword). Subpool 0 will be requested under the user's key and state. Users executing in key 0 and supervisor state will actually be assigned subpool 252. VSAM returns the address of the area in which the request parameter lists are generated in register 1, and the length of the area in register 0. You can find the length of each list by dividing the length of the area by the number of copies. You can then calculate the address of each list by using the length of each list as an offset.

If you are generating control blocks by issuing several GENCBs, specifying an area (WAREA and LENGTH parameters) for them allows you to address all of them with one base register and to avoid repetitive requests for virtual storage.

ZHYPERWRITE={YES|NO}
specifies YES to enable zHyperWrite support for the request.

Building a chain of request parameter lists

When GENCB is used to build a chain of request parameter lists, the request parameter lists may be chained using only GENCB macros or using GENCB and MODCB macros together. When only GENCB is used, the request parameter lists are created in reverse order, as follows:

```
SECOND  GENCB  BLK=RPL
        LR     2,1
FIRST   GENCB  BLK=RPL,NXTRPL=(2)
```

SECOND GENCB creates the second request parameter list, which makes its address available for the first request parameter list. The address of the request parameter list is returned in register 1 and is loaded into register 2. FIRST GENCB creates the first request parameter list and supplies the address of the next request parameter list using register notation. GENCB and MODCB macros may be used together to create a chain of request parameter lists, as follows:

```
      GENCB    BLK=RPL,COPIES=2
      LR       2,0
      SRL      2,1
      LR       3,1
      LA       4,0(2,3)
      MODCB    RPL=(3),NXTRPL=(4)
```

The GENCB macro creates two request parameter lists. The length of the parameter lists is returned in register 0 and loaded into register 2. The address of the area in which the lists were created (and, therefore, the address of the first one) is returned in register 1 and loaded into register 3. The SRL statement divides the total length of the area (register 2) by 2. The LA statement loads the address of the second request parameter list into register 4. The MODCB macro modifies the first request parameter

list (register 3) by supplying the address of the second request parameter list (register 4) in the NXTRPL parameter.

Each request parameter list in a chain should have the same OPTCD subparameters. Having different subparameters may cause logical errors. You cannot chain request parameter lists for updating or deleting records—only for retrieving records or storing new records. You cannot process records in the I/O buffer with chained request parameter lists. (OPTCD=UPD and LOC are invalid for chained request parameter lists.)

Example: GENCB macro (generate a request parameter list)

In this example, a GENCB macro is used to generate a request parameter list.

```
ACCESS  GENCB  BLK=RPL,                                              x
               ACB=ACCESS,                                           x
               AM=VSAM,                                              x
               AREA=WORK,                                            x
               AREALEN=125,                                          x
               ARG=SEARCH,                                           x
               LOC=ANY,                                              x
               MSGAREA=MESSAGE,                                      x
               MSGLEN=128,                                           x
               OPTCD=(SKP,UPD)

ACCESS  ACB    MACRF=(SKP,OUT)
WORK    DS     CL125
SEARCH  DS     CL8
MESSAGE DS     CL128
```

The GENCB macro's parameters are:

- BLK specifies a request parameter list is generated.
- ACB specifies that the request parameter list is associated with a data set and processing options identified by ACCESS.
- AREA and AREALEN specify a 125-byte work area used for processing records.
- ARG specifies the address of the search argument.
- LOC specifies that VSAM obtain storage for the request parameter list in an area above 16 megabytes.
- MSGAREA and MSGLEN specify a 128-byte area used for physical-error messages.
- OPTCD specifies the subparameters that govern the request defined by the request parameter list identified by SKP and UPD.

Example: GENCB macro (generate a request parameter list)

In this example, a GENCB macro is used to generate a request parameter list (RPL). In this example the user provides the storage to contain the RPL. Because the generate form of the macro is used, the GENCB parameter list is built in a remote area and passed to VSAM for action.

```
         LA    10,LEN2             Get length of the GENCB parameter
                                   list returned by the GENCB macro.
         GETMAIN R,LV=(10)         Get storage for the area in which
                                   the GENCB parameter list is to
                                   be built.
         LR    2,1                 Save addr of GENCB parameter-list
                                   area.
GENCB1   GENCB BLK=RPL,            One copy generated; VSAM builds    x
               ACB=ACCESS,         the RPL in the storage provided    x
               AM=VSAM,            at the location pointed to by      x
               AREA=WORK,          WAREA.                             x
               AREALEN=125,                                           x
               ARG=SEARCH,                                            x
               LENGTH=RPLLNGTH,                                       x
               MSGAREA=MESSAGE,                                       x
               MSGLEN=128,                                            x
               OPTCD=(SKP,UPD),                                       x
               WAREA=MYRPL,                                           x
               MF=(G,(2),LEN2)
               .
```

```
               .
               .
ACCESS   ACB   MACRF=(SKP,OUT)
WORK     DS    CL125
SEARCH   DS    CL8
MESSAGE  DS    CL128
         DS    0F
MYRPL    DS    CL(RPLLNGTH)          Storage in which the RPL is to be
                                     built.
ANYNAME  DSECT Avoid generation in CSECT
RPLSTART RPL   AM=VSAM
RPLEND   DS    0F
RPLLNGTH EQU   RPLEND-RPLSTART
```

The GENCB macro's parameters are:

- BLK specifies a request parameter list is generated.
- ACB specifies that the request parameter list is associated with a data set and processing options identified by ACCESS.
- AREA and AREALEN specify a 125-byte work area used for processing records.
- ARG specifies the address of the search argument.
- LENGTH specifies that the length of the storage you provide for the RPL is the value of RPLLNGTH.
- MSGAREA and MSGLEN specify a 128-byte area used for physical-error messages.
- OPTCD specifies the subparameters that govern the request defined by the request parameter list identified by SKP and UPD.
- WAREA specifies that the storage you provide for the RPL begins at label MYRPL.
- MF specifies that the GENCB parameter list is to be built in the location specified by register 2. Also, the expansion of the GENCB macro will equate LEN2 to the length of the GENCB parameter list.

GENCB—List form

The format of the list form of GENCB is:

[*label*]	GENCB	BLK={ACB|EXLST|RPL} [,AM=VSAM] [,COPIES=*abs expression*] [,*keyword*={*address*|*name*|*abs expression*|*option*},...] [,LENGTH=*abs expression*] [,LOC={BELOW|ANY}] [,RMODE31={ALL|BUFF|CB|NONE}] ,MF={L|(L,*address*[,*label*])} [,WAREA=*address*]

GENCB—Execute form

The format of the execute form of GENCB is:

[*label*]	GENCB	BLK={ACB|EXLST|RPL} [,AM=VSAM] [,COPIES=*abs expression*] [,*keyword*={*address*|*name*|*abs expression*|*option*},...] [,LENGTH=*abs expression*] [,LOC={BELOW|ANY}] [,RMODE31={ALL|BUFF|CB|NONE}] ,MF=(E,*address*) [,WAREA=*address*]

GENCB—Generate form

The format of the generate form of GENCB is:

[*label*]	GENCB	BLK={ACB|EXLST|RPL} [,AM=VSAM] [,COPIES=*abs expression*] [,*keyword=address|name|abs expression|option*},...] [,LENGTH=*abs expression*] [,LOC={BELOW|ANY}] [,RMODE31={ALL|BUFF|CB|NONE}] ,MF=(G,*address*[,*label*]) [,WAREA=*address*]

GET—Retrieve a record

Use the GET macro to retrieve a record.

When coding a GET request, spanned records that are stored in a key-sequenced data set cannot be retrieved using addressed retrieval. For more information, see Retrieving records in *z/OS DFSMS Using Data Sets*.

The format of the GET macro is:

[*label*]	GET	RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the GET macro.

RPL=*address*
specifies the address of the request parameter list that defines this GET request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Example 1: keyed-sequential retrieval—forward (KSDS, RRDS)

In this example, a GET macro is used to sequentially retrieve records by key. Retrieval is in a forward direction. Fixed-length, 100-byte records are moved to a work area. Processing is synchronous.

```
INPUT   ACB     MACRF=(KEY,      All MACRF and OPTCD subparameters         x
                SEQ.IN)          specified are defaults and could have     x
                                 been omitted.

RETRVE  RPL     ACB=INPUT,                                                 x
                AREA=IN,                                                   x
                AREALEN=100,                                               x
        .       OPTCD=(KEY,SEQ,                                            x
        .       SYN,NUP,MVE)
        .
LOOP    GET     RPL=RETRVE       This GET or identical GETs can be issued, x
                                 with no change in the RPL, to retrieve    x
                                 subsequent records in key sequence.
        LTR     15,15
        BNZ     ERROR
        .
        B       LOOP
ERROR   ...                      Request was not accepted, or failed.
        .
IN      DS      CL100            IN contains a data record after GET is    x
                                 completed.
```

The records are retrieved in key sequence in a forward direction. No search argument has to be specified; VSAM is positioned at the first record in key sequence when the data set is opened, and the next record is retrieved automatically as each GET is issued. The branch to ERROR can be taken if the end of the data set is reached.

If the data set is a variable-length RRDS, supply the record length in the RECLEN field in the RPL.

Example 2: keyed-sequential retrieval—backward (KSDS, RRDS)

This example differs from the previous one in that a POINT macro is issued to the last record in the data set and the records are retrieved in a backward direction.

```
INPUT   ACB    DDNAME=INPUT,                                          x
               EXLST=EXLST1

RETRVE  RPL    ACB=INPUT,       Define RPL for last record positioning  x
               AREA=IN,         and backward processing.                x
               AREALEN=100,                                             x
               OPTCD=(KEY,SEQ,                                          x
               LRD,BWD)

EXLST1  EXLST  EODAD=EOD        Define end of data.

        POINT  RPL=RETRVE       Position to last record (no argument
                                is required).
        LTR    15,15
        BNZ    ERROR
LOOP    GET    RPL=RETRVE       Get previous record.
        LTR    15,15
        BNZ    ERROR
        .
        B      LOOP
EOD     EQU    *                Come here for end of data.
ERROR   ...                     Request failed.
        .
IN      DS     CL100            Area for retrieved record.
```

Example 3: skip-sequential retrieval (KSDS, variable-length RRDS)

In this example, a GET macro is used to retrieve variable-length records synchronously. Records are processed in the I/O buffer. The search argument is full key, compared greater-than-or-equal; key length is 8 bytes.

The records are retrieved in key sequence, but some records are skipped. Skip-sequential retrieval is similar to keyed-direct retrieval, except that you must retrieve records in ascending sequence (with skips) rather than in a random sequence.

If the data set is a variable-length RRDS, specify the relative record number in the ARG field, and the record length in the RECLEN field in the RPL.

```
        GENCB  BLK=ACB,          VSAM gets an area in virtual storage   x
               DDNAME=INPUT,     to generate the access method control  x
               MACRF=(KEY,       block and returns the address in       x
               SKP,IN)           register 1.

        LTR    15,15
        BNZ    CHECK0
        LR     2,1
        GENCB  BLK=RPL,                                                 x
```

```
               ACB=(2),                                                 x
               AREA=RCDADDR,                                            x
               AREALEN=4,                                               x
               ARG=SRCHKEY,                                             x
               OPTCD=(KEY,SKP,                                          x
               SYN,NUP,KGE,                                             x
               FKS,LOC)
        LTR    15,15
        BNZ    CHECK0
        LR     3,1              Address of the request parameter list.
        .
LOOP    MVC    SRCHKEY,source   Search argument for retrieval, moved in
                                from a table or a transaction record.
        GET    RPL=(3)
        LTR    15,15
        BNZ    ERROR
        SHOWCB AREA=RCDLEN,     Display the length of the record.       x
               FIELDS=RECLEN,                                           x
```

```
               LENGTH=4,                                              x
               RPL=(3)

        LTR    15,15
        BNZ    CHECK0
        .
        B      LOOP
ERROR   ...                     Request was not accepted, or failed.
CHECK0  ...                     Generation or display failed.
        .
RCDADDR DS     F                Work area into which VSAM puts the     x
                                address of a data record within the    x
                                within the I/O buffer (OPTCD=LOC).

SRCHKEY DS     CL8              Search argument for retrieval.

RCDLEN  DS     F                For displaying variable record lengths.
```

The macros and instructions are as follows:

- The first GENCB generates an access method control block, which specifies keyed, skip-sequential, and input processing. The address of the access method control block is stored in register 2.
- The second GENCB generates a request parameter list. The address of the request parameter list is stored in register 3.
- MVC moves the search argument into SRCHKEY, the area defined for the search argument.
- GET specifies that the record pointed at by the request parameter list whose address is in register 3 is to be retrieved. Records are retrieved by a skip-sequential search through the sequence set of the index.

Example 4: addressed-sequential retrieval (ESDS)

In this example, one GET macro is used to retrieve multiple fixed-length, 20-byte records. The records are moved to a work area (only option).

```
BLOCK   ACB    DDNAME=INPUT,                                          x
        .      MACRF=(ADR,SEQ,                                        x
        .      IN)
        .
        GENCB  BLK=RPL,                                               x
               COPIES=10,                                             x
               ACB=BLOCK,                                             x
               OPTCD=(ADR,SEQ,                                        x
               SYN,NUP,MVE)

        LTR    15,15
        BNZ    CHECK0
        LA     3,10            Number of lists(10).
        LR     2,1             Address of the first list.

        LR     1,0             Length of all of the lists.  Registers 0 x
                               and 1 contain length and address of the  x
                               generated control blocks when VSAM       x
                               returns control after GENCB.

        SR     0,0             Prepare for following division.

        DR     0,3             Divide number of lists into length of    x
                               all the lists.

        LR     3,1             Save the resulting length of a single    x
                               list for an offset.

        LR     4,2             Save address of the first list.

        LA     5,RECAREA       Address of the first work area. Do the   x
        .                      following 6 instructions 10 times to set x
        .                      up all the request parameters lists. The x
                               10th time, register 4 must be set to 0   x
                               to indicate the last request parameter   x
                               list in the chain.

        AR     4,3             Address the next list.

        MODCB  RPL=(2),        In each request parameter list, indicate x
```

```
                NXTRPL=(4),    the address of the next list and the     x
                AREA=(5),      address and length of the work area.     x
                AREALEN=20
```

```
        LTR     15,15
        BNZ     CHECK0
        AR      2,3            Address the next list.
        LA      5,20(5)        Address the next work area. Restore      x
        .                      register 2 to address the first list     x
        .                      before continuing to process.
LOOP    GET     RPL=(2)
        LTR     15,15
        BNZ     ERROR          Process the 10 records that have been    x
        .                      retrieved by the GET.
        .
        B       LOOP
CHECK0  ...
ERROR   ...                    Display the feedback field (FIELDS=FDBK) x
                               of each request parameter list to find   x
                               out which one had an error.

RECAREA DS      CL200          Space for a work area for each of the    x
                               10 request parameter lists.
```

The GENCB macro generates 10 request parameter lists; the lists are subsequently chained together by using the MODCB macro to modify the NXTRPL parameter in each copy. Because SEQ is specified in each request parameter list, and no previous request has been issued against the access method control block since it was opened, retrieval begins at the beginning of the data set. Each time the GET macro is executed, VSAM is positioned at the next record in RBA sequence. VSAM moves each record into the work area provided for the request parameter list that identifies the record.

If an error occurs for one of the request parameter lists in the chain and you supply error-analysis routines, VSAM takes a LERAD or SYNAD exit before returning to your program. Register 15 is set to indicate the status of the request. A code of 0 indicates that no error was associated with any of the request parameter lists. Any other code indicates that an error occurred for one of the request parameter lists. You should issue a SHOWCB macro for each request parameter list in the chain to find out which had an error. VSAM does not process any of the request parameter lists except the one with an error.

Example 5: sequential retrieval for a fixed-Length RRDS

In this example, a GET macro is used to sequentially retrieve records by relative record number. Fixed-length, 100-byte records are moved to a work area. Processing is synchronous.

```
INPUT   ACB   MACRF=(KEY,SEQ)  All MACRF and OPTCD subparameters      x
                               are defaults and could be omitted.
RETRVE  RPL   ACB=INPUT,                                             x
              AREA=IN,                                               x
              AREALEN=100,                                           x
              ARG=RCDNO,                                             x
              OPTCD=(KEY,SEQ,                                        x
              SNY,NUP,MVE)
```

```
        .
LOOP    GET   RPL=RETRVE       This GET or identical GETs can be      x
                               issued, with no change in the RPL, to  x
                               retrieve subsequent records in         x
                               relative record number sequence.
        LTR   15,15
        BNZ   ERROR
        .
        B     LOOP
ERROR   ...                    Request was not accepted or it failed.
        .
IN      DS    CL100            IN contains a data record after GET is x
                               completed.
RCDNO   DS    CL4              VSAM returns relative record number of x
                               retrieved record in this field.
```

The records are retrieved in relative record number sequence. Empty records are bypassed for sequential retrieval. A 4-byte search argument must be specified. The relative record number of each record retrieved is stored in the search argument. VSAM is positioned at the first relative record when the data

set is opened, and the next not empty record is retrieved automatically as each GET is issued. The branch to ERROR is taken when the end of the data set is reached.

Example 6: keyed-direct retrieval (KSDS, RRDS)

In this example, a GET macro is used to retrieve fixed-length, 100-byte records directly by key. The key length is 15 bytes; the search argument is a 5-byte generic key, compared equal. The control blocks are generated at assembly.

```
INPUT   ACB    MACRF=(KEY,                                          x
               DIR,IN)

RETRVE  RPL    ACB=INPUT,       Specify all parameters for the request x
               AREA=IN,         in the RPL macro.                   x
               AREALEN=4,                                           x
               OPTCD=(KEY,                                          x
               DIR,SYN,NUP,                                         x
               KEQ,GEN,LOC),                                        x
               ARG=KEYAREA,                                         x
        .      KEYLEN=5
        .
LOOP    MVC    KEYAREA,SOURCE   Search argument for retrieval, moved   x
                                in from a table or a transaction       x
                                record.

        GET    RPL=RETRVE       This GET or identical GETs can be      x
                                issued with no change in the RPL:      x
                                specify each new search argument in    x
                                the field KEYAREA.
```

```
        LTR    15,15
        BNZ    ERROR
        .                       Process the record.
        .
        B      LOOP
ERROR   ...                     Request was not accepted, or failed.
        .
IN      DS     CL4              VSAM puts here the address of the      x
                                record within the I/O buffer.
KEYAREA DS     CL5              You specify the search argument here.
```

The generic key specifies a class of records. For example, if you search on the first third of employee number, VSAM positions at and retrieves the first of several records starting with the specified characters. To retrieve all the records in that class, either switch to sequential access or to a full-key search with a greater-than-or-equal comparison.

The search argument can be a key or relative record number. If the data set is a variable-length RRDS, supply the record length in the RECLEN field in the RPL.

Example 7: addressed-direct retrieval (ESDS, KSDS)

In this example, a GET macro is used to retrieve fixed-length 20-byte records. The records are to be moved to a work area.

```
BLOCK   ACB    DDNAME=INPUT,      Access method control               x
        .      MACRF=(ADR, DIR,   block generated at assembly.        x
        .      IN)
        .
        GENCB  BLK=RPL,           Request parameter list generated    x
               ARG=SRCHADR,       at execution.                       x
               AREA=IN,                                               x
               AREALEN=20,                                            x
               COPIES=1,                                              x
               ACB=BLOCK,                                             x
               OPTCD=(ADR, DIR,                                       x
               SYN, NUP, MVE)

        LTR    15,15
        BNZ    CHECK0
        LR     2, 1               Address of the list.
        .
LOOP    MVC    SRCHADR,           Search argument for retrieval;      x
```

```
                                  calculated or moved in from a table  x
                                  or a transaction record.
        GET     RPL=(2)
        LTR     15, 15
        BNZ     ERROR
        .                         Process the record.
        .

        B       LOOP
CHECK0  ...                       Generation failed.
ERROR   ...                       Request was not accepted, or failed.
        .
IN      DS      CL20              VSAM puts a record here for each     x
                                  GET request.
SRCHADR DS      CL4               You specify the RBA search argument  x
                                  here for each request.
```

The RBA provided for a search argument must match the RBA of a record. Keyed insertion and deletion of records in a key-sequenced data set will probably cause the RBAs of some records to change. Therefore, if you process a key-sequenced data set by addressed-direct access (or by addressed-sequential access using POINT), you need to keep track of changes. You can use the JRNAD exit for this purpose. See “EXLST—Generate an exit list at assembly time” on page 30.

Example 8: switch from direct to sequential retrieval

In this example, GET macros are used to retrieve fixed-length, 100-byte records. The retrieval is by means of an alternate index path defined with the non-unique key option. Every time a non-unique key is retrieved, the program switches to sequential processing to retrieve the other records with the same key. The control blocks were generated at assembly, but the MODCB macro is used to modify the request parameter list to permit switching from keyed-direct to keyed-sequential retrieval. For the direct request preceding sequential requests, the search argument is an 8-byte, generic key, compared equal. Positioning is requested for direct requests.

```
INPUT   ACB     MACRF=(KEY,DIR,   Both direct and sequential access     x
                SEQ,IN)           specified.

RETRVE  RPL     ACB=INPUT,        NSP specifies that VSAM is to        x
                AREA=IN,          remember its position.               x
                AREALEN=100,                                           x
                OPTCD=(KEY,DIR,                                        x
                SYN,NSP,KEQ,                                           x
                GEN,MVE),                                              x
                ARG=KEYAREA,                                           x
        .       KEYLEN=8
        .
LOOP    MVC     KEYAREA,source    Search argument for direct retrieval; x
                                  moved in from a table or a           x
                                  transaction.
LOOP1   GET     RPL=RETRVE
        LTR     15,15
        BNZ     ERROR
        .
        SHOWCB  RPL=RETRVE,       Extract feedback information.        x
                AREA=FDBAREA,                                          x
                FIELDS=FDBK
        LTR     15,15
        BNZ     ERROR
        CLI     ERRCD,8           Does a duplicate key follow?

        BE      SEQ               Yes; retrieve duplicates             x
                                  sequentially.

        B       LOOP              No; retrieve next record in direct   x
                                  mode.
SEQ     MODCB   RPL=RETRVE,       Alter request parameter list for     x
                OPTCD=SEQ         sequential access.
        LTR     15,15
        BNZ     CHECKO
SEQGET  GET     RPL=RETRVE        Do sequential retrieval.
        LTR     15,15             Test for error.
        BNZ     ERROR
        .
        SHOWCB  RPL=RETRVE,       Extract feedback information.        x
                AREA=FDBAREA,                                          x
```

```
                FIELDS=FDBK
        LTR     15,15
        BNZ     ERROR
        CLI     ERRCD,8           Does a duplicate key follow?
        BE      SEQGET            Yes; retrieve sequentially.
```

```
DIR     MODCB   RPL=RETRVE,       Alter request parameter list for      x
                OPTCD=DIR         direct access.
        LTR     15,15
        BNZ     CHECKO
        B       LOOP              Prepare new search argument.
ERROR   ...                       Request was not accepted, or failed.
CHECKO  ...                       Modification failed.
        .
IN      DS      CL100             VSAM puts retrieved records here.

KEYAREA DS      CL8               Specify the generic key for a direct  x
                                  request here.
FDBAREA DS      OF                Feedback area for SHOWCB.
        DS      1C                Reserved.
TYPECD  DS      1C                Error type code.
CMPCD   DS      1C                Component code.
ERRCD   DS      1C                Reason code.
```

Positioning is associated with a request parameter list; the MODCB macro modifies a single request parameter list that alternately defines requests for both types of access rather than using a different request parameter list for each type.

With direct retrieval, VSAM does not remember its position for subsequent sequential retrieval unless you explicitly request it (OPTCD=NSP or UPD). After a direct GET for update, VSAM is positioned for a subsequent PUT, ERASE, or sequential GET. If you modify OPTCD=(DIR,NUP) to OPTCD=SEQ, you must issue POINT to get VSAM positioned for sequential retrieval, as NUP indicates that no positioning is desired with a direct GET.

If you have chained many request parameter lists together, one position is remembered for the whole chain. For example, if you issue a GET that gives the address of the first request parameter list in the chain, the position of VSAM when the GET request is complete is at the record following the record defined by the last request parameter list in the chain. Therefore, modifying OPTCD=(DIR,NSP) in each request parameter list in a chain to OPTCD=SEQ implies continuing with sequential access relative to the last of the direct request parameter lists.

IDALKADD—RLS record locking

The IDALKADD macro is an RLS only VSAM request macro. It is used by applications or application support packages such as CICS® File Control that perform logging of changes to VSAM data sets. With logging, it is necessary to create a log entry before making the corresponding change to the data set or database. The log entry must uniquely identify the inserted, deleted, or changed record. Logging an ADD to a KSDS in the case where VSAM rejects the ADD due to a duplicate key condition presents a problem. Also, the record identification for an ESDS is the record RBA and logging an ADD to an ESDS implies the RBA of the record is known before actually ADDing the record. This IDALKADD request addresses these two situations.

IDALKADD to a KSDS, RRDS or VRRDS via the base or a path performs duplicate key or RRN checking. If a record with the specified key/RRN already exists in the base, the IDALKADD fails with the duplicate key/RRN error status.

The PUT request must use the same RPL as was used by the IDALKADD. The IDALKADD and PUT NUP are a request pair in the same sense as GET UPD and PUT UPD are a request pair. Reuse of the RPL before issuing the PUT NUP cancels the IDALKADD. The length of the record specified on IDALKADD and the subsequent PUT must be the same or the PUT request is rejected with an invalid record length reason code. For a KSDS, RRDS, or VRRDS, the PUT must specify a record with the same base key/RRN as was specified by the IDALKADD request.

Even though an IDALKADD is successful, the corresponding PUT NUP may fail. An example of where the PUT NUP would fail is the condition where the PUT NUP would create a duplicate key in an alternate index and the alternate index requires unique keys. In this case, the PUT NUP fails.

IDALKADD is supported for both base and path access. IDALKADD is supported for both recoverable spheres and non-recoverable spheres. It is supported for KSDSs, ESDSs, RRDSs, and VRRDSs.

The record lock acquired by an IDALKADD request is released as follows:

- Recoverable Sphere

  Only CICS transactions are allowed to add records to a recoverable sphere. The record lock is released at the end of the CICS transaction.

- Non-Recoverable Sphere

  The following events release the record lock.

  – The paired PUT NUP is issued and the data CI containing the new record has been written to DASD and the CF.
  – An ENDREQ is issued on the string.
  – The string (RPL) is re-used without issuing the paired PUT NUP.
  – The CICS transaction reaches end-of-transaction.

VSAM does not support PUT NUP,SEQ in backward processing mode. This also means IDALKADD SEQ,BWD is not supported.

The format of the IDALKADD macro is:

[*label*]	IDALKADD	RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the IDALKADD macro.

RPL=*address*
specifies the address of the request parameter list that defines this IDALKADD request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

The following RPL parameters apply to this request:

AREA
Contains a copy of the record that will be added to the data set by a PUT NUP request

When you issue IDALKADD to a KSDS, it obtains a record lock on the specified record. The record lock name is derived from the base key of the record. The base key is extracted from this copy of the record.

AREALEN
Length of the record. The subsequent PUT must specify the same length.

ARG
For an IDALKADD DIR/SKP to a RRDS or an IDALKADD DIR/SKP/SEQ request to a VRRDS, the application provides the RRN of the new record here.

MODCB—Modify an access method control block

The format of the MODCB macro used to modify an access method control block is:

[*label*]	MODCB	ACB=*address* [BSTRNO=*abs expression*] [,BUFND=*abs expression*] [,BUFNI=*abs expression*] [,BUFSP=*abs expression*] [,DDNAME=*character string*] [,EXLST=*address*] [,MACRF=([ADR][,CNV] [,KEY] [,CFX|NFX] [,DDN|DSN] [,DFR|NDF] [,DIR][,SEQ][,SKP] [,ICI|NCI] [,IN][,OUT] [,NIS|SIS] [,NRM|AIX] [,NRS|RST] [,NSR|LSR|GSR] [,NUB|UBF])] [,DB={YES|NO}] [,MAREA=*address*] [,MLEN=*abs expression*] [,PASSWD=*address*] [,RMODE31={ALL|BUFF|CB|NONE}] [,SHRPOOL=*abs expression*] [,STRNO=*abs expression*] [,ZHYPERWRITE={YES|NO}]

The subparameters of the MODCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Subparameters with GENCB, MODCB, SHOWCB, and TESTCB" on page 5, further defines these operand expressions.

label
specifies 1 to 8 characters that provide a symbolic address for the MODCB macro.

ACB=*address*
specifies the address of the access method control block to be modified. The data set identified by the access method control block must not be opened. A request to modify the access method control block of an open data set will fail.

Important: The remaining parameters represent parameters of the ACB macro that can be modified. The value specified replaces the value, if any, presently in the access method control block. *There are no defaults*. For an explanation of these parameters, see "ACB—Generate an access method control block at assembly time" on page 9.

If MODCB is used to modify a MACRF subparameter, other subparameters are unaffected, except when they are mutually exclusive. For example, if you specify MACRF=ADR in the MODCB and MACRF=KEY is already indicated in the control block, both ADR and KEY are now indicated. But, if you specify MACRF=UBF in the MODCB and NUB is indicated, only UBF will now be indicated.

The RMODE31 parameter tells the VSAM OPEN routines where to obtain storage for the control blocks and I/O buffers. Therefore, the only time the values specified by the RMODE31 parameter have any effect on VSAM is on the setting just before an OPEN is issued. At other times, changing these values has no effect on the residency of the control blocks and I/O buffers. RMODE31 is ignored for RLS processing.

If MODCB RPL is used to change the address of an ACB, you must first issue an ENDREQ macro.

Restriction: If you issue a MODCB for a non-VSAM and non-VTAM ACB, the results will be unpredictable.

Example: MODCB macro (modify an access method control block)

In this example, a MODCB macro is used to modify the name of the exit list in an access method control block.

```
        MODCB  ACB=BLOCK,          BLOCK was generated at              x
               EXLST=EGRESS        assembly.
```

MODCB—Modify an exit list

The format of the MODCB macro used to modify an exit list is:

[*label*]	MODCB	EXLST=*address* [,EODAD=([*address*][,A|N][,L])] [,JRNAD=([*address*][,A|N][,L])]: [,LERAD=([*address*][,A|N][,L])] [,SYNAD=([*address*][,A|N][,L])]

The subparameters of the MODCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. “Subparameters with GENCB, MODCB, SHOWCB, and TESTCB” on page 5, further defines these operand expressions.

See: *z/OS DFSMS Using Data Sets* for information about what determines the addressing mode and the parameter list residency mode set when the exit routine gets control.

label
specifies 1 to 8 characters that provide a symbolic address for the MODCB macro.

EXLST=*address*
specifies the address of the exit list to be modified. You can modify an exit list at any time—that is, before or after opening the data sets for which the list indicates exit routines. You cannot, however, add an entry to the exit list if it changes the exit list's length; the exit list must already be large enough to contain the new exit address. The order in which addresses are stored in the EXLST control block is: EODAD, SYNAD, LERAD, JRNAD, and UPAD. For example, if you generate an exit list with only the LERAD exit, you can add entries for EODAD and SYNAD later. However, you cannot add the JRNAD exit address, because doing so would increase the size of the EXLST control block. The MODCB macro does not support the UPAD user exit.

The remaining parameters represent parameters of the EXLST macro that can be modified or added to an exit list. For an explanation of these parameters, see “EXLST—Generate an exit list at assembly time” on page 30.

Requirement: If the JRNAD exit is changed for an OPEN ACB, then the ACB must be closed and reopened to use the modified JRNAD exit.

For more information about user exit routines, see *z/OS DFSMS Using Data Sets*.

Example: MODCB macro (modify an exit list)

In this example, a MODCB macro is used to activate an exit in an exit list.

```
          MODCB EXLST=(*,        Indirect notation is used to specify   x
                EXLSTADR),       the address of the exit list generated x
          .     EODAD=(EOD,L,A)  at execution.
          .
EOD       DC    C'ENDUP'
EXLSTADR DS     F                When the exit list was generated,      x
                                 its address was saved here.
```

The MODCB macro's parameters are:

- EXLST specifies the address of the exit list being modified is located at EXLSTADR.
- EODAD specifies the entry for the end-of-data routine is marked active in the exit list that has an address at EXLSTADR. The name of the end-of-data routine (ENDUP) is at EOD.

MODCB—Modify a request parameter list

The format of a MODCB macro used to modify a request parameter list is:

[*label*]	MODCB	RPL=*address* [,ACB=*address*] [,AREA=*address*] [,AREALEN=*abs expression*] [,ARG=*address*] [,ECB=*address*] [,KEYLEN=*abs expression*] [,MSGAREA=*address*] [,MSGLEN=*abs expression*] [,NXTRPL=*address*] [,OPTCD=([ADR|CNV|KEY] [,DIR|SEQ|SKP] [,ARD|FRD|LRD] [,FWD|BWD] [,ASY|SYN] [,NSP|NUP|UPD] [,KEQ|KGE] [,FKS|GEN] [,LOC|MVE]] [,ARA31|ARA64] [,RBA|XRBA])] [,RECLEN=*abs expression*] [,DBARGLN=*abs expression* [,TRANSID=*abs expression*] [,ZHYPERWRITE={YES|NO}]

The subparameters of the MODCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Subparameters with GENCB, MODCB, SHOWCB, and TESTCB" on page 5, further defines these operand expressions.

label
specifies 1 to 8 characters that provide a symbolic address for the MODCB macro.

RPL=*address*
specifies the address of the request parameter list being modified. You may not modify an active request parameter list; one that defines a request that has been issued but not completed. To modify such a request parameter list, you must first issue a CHECK or an ENDREQ macro.

Important: If you use MODCB to modify fields in the RPL, you must first disconnect the RPL from any process by issuing an ENDREQ macro.

The remaining parameters represent parameters of the RPL macro that can be modified. The value specified replaces the value, if any, presently in the request parameter list. *There are no defaults*. For an explanation of these parameters, see "GENCB—Generate a request parameter list at execution time" on page 40.

If MODCB is used to modify an OPTCD subparameter within a group of subparameters, the current subparameter for that group is changed because only one subparameter in a group is effective at a time. Only the specified OPTCD subparameter is changed.

Example: MODCB macro (modify a request parameter list)

In this example, a MODCB macro is used to modify the record length field in a request parameter list.

This example shows the one exception to GENCB, MODCB, SHOWCB, and TESTCB building a parameter list and passing it to the control block manipulation module in register 1. The RPL address (in register 2) is loaded into register 1 and the RECLEN value (in register 3) is loaded into register 0. These registers are passed to the control block manipulation macro. This occurs when the LIST, EXECUTE, or GENERATE form of the MODCB macro is not used and the only parameter specified other than RPL, is RECLEN.

```
        L       3,length        Load the new record length.

        MODCB   RPL=(2),        Register 2 contains the address           x
                                of the request parameter list.            x
                RECLEN=(3)      Register 3 contains the record length.
```

The MODCB macro's parameters are:

- RPL specifies register 2 contains the address of the request parameter list being modified.
- RECLEN specifies the record length field is being modified. The contents of register 3 replace the current value in the RECLEN field.

MODCB—List form

The format of the list form of MODCB is:

[*label*]	MODCB	{ACB|EXLST|RPL}=*address* ,*keyword*={*address*|*name*|*abs expression*|*option*},... ,MF={L|(L,*address*[,*label*])}

MODCB—Execute form

The format of the execute form of MODCB is:

[*label*]	MODCB	[{ACB|EXLST|RPL}=*address*] ,*keyword*={*address*|*name*|*abs expression*|*option*},... ,MF=(E,*address*)

Requirement: If the execute form of MODCB is used and EXLST is used as a keyword to be processed, the block must be identified by ACB=.

MODCB—Generate form

The format of the generate form of MODCB is:

[*label*]	MODCB	{ACB|EXLST|RPL}=*address* ,*keyword*={*address*|*name*|*abs expression*|*option*},... ,MF=(G,*address*[,*label*])

MRKBFR—Mark buffer

If you are using local or global shared resources, use the MRKBFR macro to mark a buffer.

The format of the MRKBFR macro is:

[*label*]	MRKBFR	MARK={DINVALID|XINVALID|OUT|RLS} ,RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the MRKBFR macro.

MARK={DINVALID|XINVALID|OUT|RLS}
specifies the buffer identified in the RPL is either marked for output, or is to be released from either exclusive control or shared status. To do both, issue MRKBFR twice: once with MARK=OUT, once with MARK=RLS.

DINVALID|XINVALID
specifies that either the data component or the index component buffers are to be marked invalid. The buffers being invalidated are those that contain records with RBA values within the RBA range pointed to by the RPL ARG address. DINVALID specifies that the data component buffers be marked invalid. XINVALID specifies that the index component buffers be marked invalid.

OUT
specifies that the buffer be marked for output. The buffer is kept either under exclusive control or in shared status.

RLS
specifies that the buffer be released either from exclusive control or shared status.

RPL=*address*
specifies the address of the request parameter list defining the MRKBFR request. Use the SCHBFR or GET RPL to locate the buffer being marked or released. These RPL parameters have meaning for MRKBFR:

ACB=*address*

ARG=*address*
The address of the 8-byte field that contains the beginning and ending RBAs of the range being searched on.

For compressed data sets, the RBA of another record or the address of the next record in a buffer cannot be determined using the length of the current record or the length of the record provided to VSAM.

For extended addressing, the addressof a 16-byte field containing the beginning and ending 8-byte RBAs of the range.

ECB=*address*

TRANSID=*number*

All other RPL parameters are ignored. RPLs are assumed not to be chained. OPTCD=LOC is assumed.

If the ACB related to the RPL has MACRF=GSR, the program issuing MRKBFR must be in supervisor state with protection key 0 to 7.

OPEN—Connect program and data

Use the OPEN macro to open a data set.

The format of the OPEN macro is:

[*label*]	OPEN	(*address*[(*options*)][,...]) [,MODE={24|31}]

label
specifies 1 to 8 characters that provide a symbolic address for the OPEN macro.

address
specifies the address of the ACB or DCB for the data sets being opened. You may specify the address either in register notation (using a register from 2 through 12, in parentheses) or with an expression that generates a valid relocatable A-type address constant. If you use register notation to open one data set, enclose the expression identifying the register within two sets of parentheses: OPEN ((2)).

options
specifies options parameters used only in opening non-VSAM data sets. VSAM ignores options specified with the address of an access method control block.

Because the OPEN parameters are positional, if options are not specified, you must insert a comma before coding a subsequent parameter.

MODE =
specifies the format of the OPEN parameter list being generated.

24
specifies that a standard form (24-bit) parameter list address be generated. The parameter list must reside below 16 megabytes and point to an ACB residing below 16 megabytes.

31
specifies that a long form (31-bit) parameter list address be generated. The parameter list can reside above 16 MB and can point to an ACB residing above 16 MB. This parameter value must be coded if the parameter list or the VSAM/VTAM ACB resides above 16 megabytes.

Rule: For non-RLS, if the VSAM control blocks and buffers are to reside above 16 megabytes, the RMODE31 parameter must be specified in the ACB before the OPEN is issued.

Note: You can put more than one ACB/DCB in an OPEN/CLOSE macro or you can include BOTH ACBs and DCBs. If multiple ACBs/DCBs are provided, data areas associated with each entry will not be available for reference until the entire OPEN/CLOSE is complete.

Example 1: OPEN macro used to open two data sets

In this example, the access method control block for one data set is generated at execution; the other is generated at assembly.

```
         GENCB  BLK=ACB,        An access method control block.         x
                DDNAME=DATA
         LTR    15,15

         BNZ    ERROR

         LR     2,1             Address of the control block.
```

```
         OPEN   (BLOCK,,(2))    A label is used for the access method   x
                                control block generated by ACB;         x
                                register notation is used for the       x
                                one generated by GENCB.  The two commas x
                                indicate the omission of options.

BLOCK    ACB    ,               Another access method control block.
```

Example 2: OPEN macro with a parameter list above 16 megabytes

In this example, a program is opened with a parameter list that may reside above 16 megabytes.

```
OPLSTA   OPEN   MODE=31,                                               x
                MF=(E,OPLSTB)

OPLSTB   OPEN   (ACB1,,ACB2),                                          x
                MODE=31,                                               x
                MF=L
```

Since MODE=31 is coded in the list form of the OPEN macro, VSAM ACBs and the OPEN parameter list may reside above 16 megabytes.

Rule You must maintain consistency while using the MODE operand in the MF=L and MF=E versions of the OPEN macro. If MODE=31 is specified in the MF=L version, then MODE=31 must also be coded in the corresponding MF=E version of the macro. Unpredictable results might occur if this rule is not followed.

MF=E and MF=L are not required. OPEN (ACB1),MODE=31 is also valid.

POINT—Position for access

Use the POINT macro to position a record.

The format of the POINT macro is:

[*label*]	POINT	RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the POINT macro.

RPL=*address*
specifies the address of the request parameter list defining the request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Example: position with POINT

In this example, the POINT macro is used to position at a record identified by a full key (5-byte) search argument, compared equal.

```
BLOCK    ACB   DDNAME=IO        Default MACRF subparameters sufficient.

POSITION RPL   ACB=BLOCK,       ARG parameter and KEQ and FKS OPTCD    x
               AREA=WORK,       subparameters define the POINT         x
               AREALEN=50,      request.                               x
               ARG=SRCHKEY,                                            x
         .     OPTCD=(KEY,SEQ,SYN,KEQ,FKS)
         .
LOOP     MVC   SRCHKEY,source   Search argument for positioning, moved x
                                from a table or transaction record.
         POINT RPL=POSITION
         LTR   15,15
         BNZ   ERROR
LOOP1    GET   RPL=POSITION
         LTR   15,15
         BNZ   ERROR
```

Process the record. Decide whether to skip to another position (forward or backward).

```
         BE    LOOP              Yes; skip.
         B     LOOP1             No; continue in consecutive sequence.
ERROR    ...                     Request was not accepted, or failed.
         .
SRCHKEY  DS    CL5               Search argument for positioning.

WORK     DS    CL50              VSAM puts a record here for each GET  x
                                 request.
```

PUT—Write a record

Use the PUT macro to write (load) records to an empty data set, and insert or update records into an existing data set.

To do a PUT UPD you must first do a GET UPD for the record.

The format of the PUT macro is:

[*label*]	PUT	RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the PUT macro.

RPL=*address*
specifies the address of the request parameter list defining the request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

Rule: If the PUT macro is used to load records into an empty data set, the STRNO value in the access method control block must be 1, and RPL OPTCD=DIR must not be specified. However, for an empty RRDS, DIR is allowed.

Example 1: keyed-sequential insertion (KSDS, variable-length RRDS)

In this example, a PUT macro is used to perform keyed-sequential insertion in a key-sequenced data set or variable-length RRDS. Variable-length records with a key length of 15 bytes are moved from a work area. Some records are inserted between existing records; other records are added at the end of the data set.

```
BLOCK    ACB   DDNAME=OUTPUT,                                        x
               MACRF=(KEY,SEQ,OUT)

LIST     RPL   ACB=BLOCK,                                           x
               AREA=BUILDRCD,                                        x
               AREALEN=250,                                          x
         .     OPTCD=(KEY,SEQ,                                       x
         .     SYN,NUP,MVE)
         .
LOOP     L     2,source          Put length of record to be inserted   x
                                 into register.

         MODCB RPL=LIST,         Indicate record length in request     x
               RECLEN=(2)        parameter list.

         LTR   15,15
         BNZ   CHECKO
         PUT   RPL=LIST
         LTR   15,15
         BNZ   ERROR
         B     LOOP
CHECKO   ...                     Modification failed.
ERROR    ...                     Request was not accepted, or failed.
```

```
BUILDRCD DS    CL250             Work area for building records.
```

The request parameter list, LIST, is associated with the access method control block, BLOCK. The length of each record to be inserted is put into register 2, which is subsequently used by MODCB to change the record length in the request parameter list. The record length is, therefore, correctly indicated in the request parameter list before the PUT macro is issued. The execution of the PUT macro causes VSAM to skip ahead (never back) to the next record.

Example 2: recording RBAs when loading a KSDS

In this example, a PUT macro is used to record the RBAs of records as they are loaded into a key-sequenced data set. The RBAs are recorded in a table with 20-byte entries (4 bytes for RBA, 15 bytes for associated key, and 1 byte of padding so the next entry begins on a fullword boundary).

```
         LA    3,RBATBLE         Address of the beginning of the table.
         ...
LOOP     L     2,source          Put length of record to be inserted    x
                                 into register 2.

         MODCB RPL=LIST,         Indicate record length in request      x
               RECLEN=(2)        parameter list.
```

```
        LTR    15,15
        BNZ    CHECKO
        PUT    RPL=LIST
        LTR    15,15
        BNZ    ERROR

        SHOWCB AREA=(3),        Each SHOWCB puts a record's RBA into   x
               FIELDS=RBA,      the table.                              x
               LENGTH=4,                                               x
               RPL=LIST

        LTR    15,15
        BNZ    CHECKO
        MVC    4(15,3),         Put the record's key field in the      x
               keyfield         table.

        LA     3,20(3)          Point to the next entry.
        B      LOOP
ERROR   ...                     Request was not accepted, or failed.

CHECKO  ...                     Modification or display failed.
        .
        DSECT                   Get enough virtual storage for as many x
                                table entries as there are records in  x
                                the data set.
```

```
RBATBLE DS     OF
RBA     DS     CL4
KEY     DS     CL15
        DS     CL1              Padding to keep each RBA entry on a    x
                                fullword boundary: SHOWCB's            x
                                display area must be on a fullword     x
                                boundary.
```

The need to process a key-sequenced data set by address is unusual, but by recording the RBA of each record in a key-sequenced data set, you have search arguments for possible processing of the data set by addressed-direct retrieval and by addressed-sequential retrieval using the POINT macro. (You do not need to know RBAs to process a key-sequenced data set by simple addressed-sequential retrieval, since you go from the beginning without any skips.)

You can display the RBA of a record after you issue a GET or a POINT, as well as after you issue a PUT.

Example 3: loading a fixed-length RRDS (skip-sequential and direct processing)

In this example, a PUT macro is used to store twenty 100-byte records in slots 5, 10, 15,...,100 of the data set. MODCB is used to switch to direct processing, and PUT is used to store records in slots 26 and 51 of the data set.

```
OUTACB   ACB   MACRF=(SKP,OUT,                                          x
         .     DIR,KEY)
         .
         GENCB BLK=RPL,          Generate a request parameter list      x
               ACB=OUTACB,       at execution time.                     x
               AREA=WORK,                                               x
               AREALEN=100,                                             x
               ARG=RCDNO,                                               x
               OPTCD=(KEY,SKP)

         LTR   15,15
         BNZ   GENFAIL
         LR    5,0               Save length of RPL.
         LR    6,1               Save address of RPL.
         LA    7,5               Initialize increment value.
         ST    7,RCDNO           Initialize argument to slot 5.
         LA    10,20             Initialize loop counter.
LOOP     ...                     Move new record into work.
         PUT    RPL=(6)          Store record.
         LTR    15,15
```

```
         BNZ    PUTERR           Request was not accepted, or failed.
         L      1,RCDNO
```

```
         AR     1,7
         ST     1,RCDNO          Increment argument by 5.
         BCT    10,LOOP
         MODCB  RPL=(6),         Switch to direct processing to store   x
                OPTCD=(DIR,KEY)  records in slots 51 and 26.
```

```
         LTR    15,15
         BNZ    GENFAIL
         LA     7,51
         ST     7,RCDNO          Initialize argument to slot 51.
         ...                     Move new record into WORK.
         PUT    RPL=(6)          Store record in slot 51.
         LTR    15,15
         BNZ    PUTERR           Request was not accepted, or failed.
         LA     7,26
         ST     7,RCDNO          Initialize argument to slot 26.
         ...                     Move new record into WORK.
         ...
         PUT    RPL=(6)          Store record in slot 26.
         LTR    15,15
         BNZ    PUTERR           Request was not accepted, or failed.
         B      RETURN
GENFAIL  ...                     Generation or modification failed.

PUTERR   ...                     PUT request was not accepted, or failed.

RETURN   ...                     Terminate program.
```

```
WORK     DS     CL100            100-byte work area that contains       x
                                 record to be stored by PUT macro.

RCDNO    DS     CL4              4-byte relative record number.
```

Both skip-sequential and direct processing can be used to allocate a fixed-length RRDS. The ACB is opened for output. The 4-byte search argument (RCDNO) indicates the slot number where the record is to be stored.

Example 4: keyed-sequential insertion (fixed-length RRDS)

In this example, a PUT macro is used to insert twenty 100-byte records into empty slots of a previously loaded fixed-length RRDS. If the slot is empty when the PUT is issued, the record is stored and the slot number (returned in the argument field) is stored in a table. If the slot is not empty when the PUT is issued, a duplicate record error indication is returned. When a duplicate record is indicated, the PUT is reissued until the record is successfully stored in an empty slot in the data set.

```
OUTACB   ACB    MACRF=(KEY,SEQ,                                        x
         .      OUT)
         .
         GENCB  BLK=RPL,         Generate a request parameter list.     x
                ACB=OUTACB,                                             x
                AREA=WORK,                                              x
                AREALEN=100,                                            x
                ARG=RCDNO,                                              x
                OPTCD=(KEY,SEQ)

         LTR    15,15
         BNZ    GENERR
         LR     6,1              Save the address of the RPL.
         LA     4,RRNTBLE+80     Initialize address of end of table.

         LA     3,RRNTBLE        Initialize index to relative record    x
                                 number table.
WRITERCD ...                     Move record into work area.
          .
          .
         PUT    RPL=(6)
         LTR    15,15
         BZ     STRCDNO          Branch, if PUT is successful.
         LA     10,8
         CLR    10,15            Test for logical error.
         BNE    PUTERR
         TESTCB RPL=(6),FDBK=8,  Test for duplicate record.             x
                ERET=TESTERR

         BE     WRITERCD         Branch, if duplicate record, and try   x
```

```
                                to store record in next slot.
        B       PUTERR
STRDCNO         ...
        MVC     0(4,3)RCDNO     Store relative record number in        x
                                RRNTABLE.
        LA      3,4(3)          Increment to next table entry.
        CLR     3,4
        BE      RETURN          If table full, return to caller.
        B       WRITERCD        Write next record.

GENERR  ...                     Error routine for GENCB macro.

TESTERR ...                     Error routine for TESTCB macro.
```

```
PUTERR  ...                     Error routine for PUT macro.

RETURN  ...                     Return to caller or terminate program.

RCDNO   DS      CL4             4-byte relative record number          x
                                (argument) field.
RRNTBLE DS      20F             Relative record number table.

WORK    DS      CL100           100-byte work area that contains       x
                                record to be stored by PUT macro.
```

Each record is stored in the next available slot in the data set. When a record is successfully stored, its relative record number is recorded in a table.

Example 5: skip-sequential insertion (KSDS, variable-length RRDS)

In this example, one PUT macro is used to insert multiple fixed-length, 100-byte records. Records are to be moved asynchronously from a work area.

```
OUTPUT   ACB   MACRF=(KEY,SKP,                                        x
         .     OUT)
         .
         GENCB BLK=RPL,          Generate 5 request parameter lists    x
               COPIES=5,         at execution.                         x
               ACB=OUTPUT,                                             x
               AREALEN=100,                                            x
               OPTCD=(KEY,SKP,                                         x
               ASY,NUP,MVE),                                           x
               RECLEN=100

         LTR   15,15
         BNZ   CHECKO
```

Calculate length of each list and use register notation with the MODCB macro to complete each list:

```
         MODCB RPL=(2),                                                x
               AREA=(3),                                               x
               NXTRPL=(4)
```

```
         LTR   15,15
         BNZ   CHECKO
```

Increase the value in each register and repeat the MODCB until all 5 request parameter lists have been completed. The last time, register 4 must be set to 0:

```
          .
LOOP     ...                    Restore address of first list in       x
                                register 2. Build 5 records in WORK.

         PUT   RPL=(2)          Register 2 points to the first RPL in  x
                                the chain.  The 5 records in WORK      x
                                are stored with this one PUT request.
         LTR   15,15
         BNZ   NOTACCEP
         .
         CHECK RPL=(2)
         LTR   15,15
         BNZ   ERRO
         B     LOOP
CHECKO   ...                    Generation or modification failed.
```

```
NOTACCEP ...

ERROR    ...                  Display the feedback field in each     x
                              RPL to determine which one had error.

WORK     DS    CL500          Contains five 100-byte work areas.
```

You give no search argument for storage: VSAM knows the position of the key field in each record and extracts the key from it. Skip-sequential insertion differs from keyed-direct insertion in the sequence in which records may be inserted (ascending non-consecutive sequence versus random sequence) and in performance.

With skip-sequential insertion, if you insert two or more records into a control interval, VSAM does not write the contents of the buffer to direct-access storage until you have inserted all the records. With direct insertion, VSAM writes the contents of the buffer after you have inserted each record.

Example 6: keyed-direct insertion (KSDS, RRDS)

In this example, a PUT macro is used to move fixed-length, 100-byte records from a work area.

```
OUTPUT   ACB   MACRF=(KEY,DIR,                                        x
               OUT)

DIRECT   RPL   ACB=OUTPUT,                                           x
               AREA=WORK,                                            x
               AREALEN=100,                                          x
               OPTCD=(KEY,DIR,                                       x
               ASY,NUP,MVE),                                         x
         .     RECLEN=100
         .
LOOP     PUT   RPL=DIRECT
         LTR   15,15
         BNZ   NOTACCEP
         ...
         CHECK RPL=DIRECT
         LTR   15,15
         BNZ   ERROR
         B     LOOP
NOTACCEP ...                    Request was not accepted.
ERROR    ...                    Request failed.
         .
WORK     DS    CL100            Work area.
```

The macros are as follows:

- ACB specifies the data set, OUTPUT, into which records are to be inserted, is opened for keyed-direct, output processing.
- RPL specifies the record to be inserted into the OUTPUT data set resides in a 100-byte area, WORK.

VSAM extracts the relative record number or key from the key field of each record found at WORK. Using keyed-direct access is similar to using skip-sequential access.

Example 7: addressed-sequential addition (ESDS)

In this example, a PUT macro is used to add variable-length records to a data set. The data set is assumed to be an entry-sequenced data set, because records cannot be inserted into or added to a KSDS with addressed access.

```
BLOCK    ACB    MACRF=(ADR,SEQ,                                       x
                OUT)

LIST     RPL    ACB=BLOCK,                                            x
                AREA=NEWRCD,                                          x
                AREALEN=100,                                          x
                OPTCD=(ADR,SEQ,                                       x
         .      SYN,MVE)
         .
LOOP     ...                    Build the record.
         L      3,source         Put length of record into register 3.
```

```
        MODCB  RPL=LIST,         Indicate length of new record.        x
               RECLEN=(3)

        LTR    15,15
        BNZ    CHECKO
        PUT    RPL=LIST
        LTR    15,15
        BNZ    ERROR
        B      LOOP
CHECKO  ...                     Modification failed.
ERROR   ...                     Request was not accepted, or failed.
        .
NEWRCD  DS     CL100            Build record in this work area.
```

Each record is stored in the next position after the last record in the data set. You do not have to specify an RBA or do any explicit positioning (with the POINT macro). Addressed addition of records is identical to loading a data set: when additional space is required, VSAM extends the data set.

The only difference between addressed-sequential and addressed-direct addition is when the buffers are written to external storage. The buffer is written to external storage only when it is full for sequential addition; it is written after each record for direct addition. You cannot use direct storage to load records into a data set for the first time; you must use sequential storage.

Example 8: keyed-sequential update (KSDS, RRDS)

In this example, GET and PUT macros are used to retrieve and update fixed-length, 50-byte records. Records are updated synchronously in a work area. This example requires the use of a work area because you cannot update a record in the I/O buffer.

```
UPDATA  ACB    MACRF=(KEY,SEQ,                                       x
               OUT)

LIST    RPL    ACB=UPDATA,      UPD indicates the record may be stored x
               AREA=WORK,       back (or deleted).                    x
               AREALEN=50,                                            x
               OPTCD=(KEY,SEQ,                                        x
        .      SYN,UPD,MVE)
        .
LOOP    GET    RPL=LIST

        LTR    15,15
        BNZ    ERROR
```

Decide whether to update the record.

```
        BE     LOOP             Do not update it; retrieve another.
```

Update the record.

```
        PUT    RPL=LIST         Store the record back.
        LTR    15,15
        BNZ    ERROR
        B      LOOP
ERROR   ...                     Request was not accepted, or failed.
        .

WORK    DS     CL50             VSAM puts the retrieved record here.
```

A GET for update (OPTCD=UPD) must precede a PUT for update. Besides retrieving the record to be updated, GET positions VSAM at the record retrieved, in anticipation of the succeeding update (or deletion). It is not necessary for you to store back (or delete) the record you retrieved for update. VSAM's position at the record previously retrieved allows you to issue another GET to retrieve the following record. You cannot, however, store back the previous record: the position for update has been forgotten because of the following GET.

Example 9: keyed-direct update (KSDS, variable-length RRDS)

In this example, GET and PUT macros are used to retrieve and update records. The MODCB macro is used to modify record length (RECLEN) in the request parameter list when an update causes the record

length to change. The maximum record length is 120 bytes. The search argument is a full key (5 bytes), compared equal.

```
INPUT   ACB    MACRF=(KEY,DIR,                                    x
               OUT)

UPDTE   RPL    ACB=INPUT,       UPD indicates the record may be   x
               AREA=IN,         stored back (or deleted).         x
               AREALEN=120,                                       x
               OPTCD=(KEY,DIR,                                    x
               SYN,UPD,KEQ,                                       x
               FKS,MVE),                                          x
               ARG=KEYAREA,                                       x
        .      KEYLEN=5
        .
```

Process input and get search argument into KEYAREA; proceed to retrieve a record:

```
LOOP    GET    RPL=UPDTE
        LTR    15,15
        BNZ    ERROR

        SHOWCB RPL=UPDTE,       Display the length of the record.  x
               AREA=RLNGTH,                                        x
               FIELDS=RECLEN,                                      x
               LENGTH=4

        LTR    15,15
        BNZ    CHECKO
```

Update the record. Does the update change the record's length?

```
        BE     STORE            No; length not changed.

        L      5,length         Yes; load new length into register 5.

        MODCB  RPL=UPDTE,       Modify length indication in the request x
               RECLEN=(5)       parameter list.

        LTR    15,15
        BNZ    CHECKO
STORE   PUT    RPL=UPDTE
        LTR    15,15
        BNZ    ERROR
        B      LOOP
```

```
ERROR   ...                   Request was not accepted, or failed.

CHECKO  ...                   Display or modification failed.
        .
IN      DS     CL120          Work area for retrieving, updating,     x
                              and storing a record.
KEYAREA DS     CL5            Search argument for retrieving a        x
                              record.
RLNGTH  DS     F              Area for displaying the length of a     x
                              retrieved record.
```

You cannot update records in the I/O buffer. A direct GET for update positions VSAM at the record retrieved, in anticipation of storing back (or deleting) the record. This positioning also allows you to switch to sequential access to retrieve the next record. VSAM releases exclusive control of a control interval when a PUT DIR is issued following a GET UPD request.

You do not have to store back a record that you retrieve for update, but, if you do not store it back before another retrieval, the current updates are lost.

Example 10: addressed-sequential update (ESDS)

In this example, GET and PUT macros are used to retrieve and update records in an entry-sequenced data set. The records are variable in length, a maximum of 200 bytes. The lengths of the records are not changed by update (the length of a record can never be changed by addressed access).

```
ENTRY   ACB    MACRF=(ADR,SEQ,OUT)

ADRUPD  RPL    ACB=ENTRY,         UPD indicates update (or deletion).  x
               AREA=WORK,                                              x
               AREALEN=200,                                            x
               OPTCD=(ADR,SEQ,                                         x
        .      SYN,UPD,MVE)
        .
LOOP    GET    RPL=ADRUPD
        LTR    15,15
        BNZ    ERROR

        SHOWCB RPL=ADRUPD,        Determine record length.             x
               AREA=RECLEN,                                            x
               FIELDS=RECLEN,                                          x
               LENGTH=4

        LTR    15,15
        BNZ    CHECKO
        .
```

```
        PUT    RPL=ADRUPD
        LTR    15,15
        BNZ    ERROR
        B      LOOP
ERROR   ...                       Request was not accepted, or failed.
CHECKO  ...                       Display failed.
        .
```

```
WORK    DS     CL200              Record-processing work area.
RLNGTH  DS     F                  Display area for length of records.
```

If you have inactive records in your entry-sequenced data set, you may reuse the space they occupy by retrieving the records for update and restoring a new record in their place.

With a key-sequenced data set, it is not possible to change the length of records by addressed update because the index is not used and VSAM could not split a control interval if required because of changing record length.

Addressed-direct update varies from sequential update in the specification of an RBA for a search argument.

Example 11: marking records inactive (ESDS)

In this example, GET and PUT macros retrieve a record from an entry-sequenced data set and mark it as inactive by putting a hexadecimal X'FF' in the first byte of a record. The inactive record can only be sequentially retrieved for update.

```
ENTRYSEQ ACB   MACRF=(ADR,DIR,                                         x
               OUT)

LIST     RPL   ACB=ENTRYSEQ,      UPD indicates update; storing the    x
               AREA=RECORD,       record back marked inactive.         x
               AREALEN=100,                                            x
               OPTCD=(ADR,DIR,                                         x
               SYN,UPD,MVE),                                           x
         .     ARG=RBAAREA
         .
LOOP     GET   RPL=LIST
         LTR   15,15
         BNZ   ERROR
```

Decide whether you still want the data in the record.

```
         BE    LOOP               Yes; retrieve the next record.

         MVI   RECORD,X'FF'       No; flag the record inactive.
```

```
         PUT   RPL=LIST           For an entry-sequenced data          x
                                  set, storing the record with         x
                                  an inactive indicator is             x
                                  equivalent to  deletion.
```

```
         LTR   15,15
         BNZ   ERROR
         B     LOOP
ERROR    ...                    Request was not accepted, or failed.
RECORD   DS    CL100              Work area for marking records.
RBAAREA  DS    F                  Search argument for retrieving record.
```

You cannot delete an entry-sequenced data set record. You can mark an ESDS record inactive by placing a unique flag in a conventional part of the record so that when the record is subsequently retrieved, the flag causes the record to be bypassed. To reuse the space occupied by an inactive ESDS record, retrieve it for update and store a new record in its place.

RPL—Generate a request parameter list at assembly time

Use the RPL macro to generate a request parameter list. Values for RPL macro subparameters can be specified as absolute numeric expressions, character strings, codes, and expressions that generate valid relocatable A-type address constants.

The format of the RPL macro is:

[*label*]	RPL	[ACB=*address*] [,AM=VSAM] [,AREA=*address*] [,AREALEN=*abs expression*] [,ARG=*address*] [,ECB=*address*] [,KEYLEN=*abs expression*] [,TIMEOUT=*number*] [,MSGAREA=*address*] [,MSGLEN=*abs expression*] [,NXTRPL=*address*] [,OPTCD=([ADR|CNV|KEY] [,DIR|SEQ|SKP] [,ARD|FRD]|LRD [,FWD|BWD] [,ASY|SYN] [,NSP|NUP|UPD] [,KEQ|KGE] [,FKS|GEN] [,NWAITX|WAITX] [,LOC|MVE] [,NRI|CR] [,ARA31|ARA64] [,RBA|XRBA])] [,RECLEN=*abs expression*] [,DBARGLN=*abs expression*] [,TRANSID=*abs expression*] [,ZHYPERWRITE={YES|NO}]

label
specifies 1 to 8 characters that provide a symbolic address for the generated request parameter list. You can use *label* in the request macros to give the address of the list. You can use *label* in the NXTRPL parameter of the RPL macro, when you are chaining request parameter lists, to indicate the next list.

ACB=*address*
specifies the address of the access method control block identifying the data set to which access is requested. If you used the ACB macro to generate the control block, you may specify the label of that

macro for the address. If the ACB parameter is not coded, you must specify the address before issuing the request.

AM=VSAM
specifies the access method using the control block is VSAM.

AREA=*address*
specifies the address of a work area to and from which VSAM moves a data record if you request it to do so (with the RPL parameter OPTCD=MVE). If your request is to process records in the I/O buffer (OPTCD=LOC), VSAM puts into this work area the address of a data record within the I/O buffer.

AREALEN=*abs expression*
specifies the length, in bytes, of the work area whose address is specified by the AREA parameter. Its minimum for OPTCD=MVE is the size of a data record (of the largest data record, for a data set with records of variable length). For OPTCD=LOC, the area should be 4 bytes to contain the address of a data record within the I/O buffer.

ARG=*address*
specifies the address of a field that contains the search argument for direct retrieval, skip-sequential retrieval, and positioning. For a RRDS, the ARG field must be 4 bytes long. For direct or skip-sequential processing, this field contains your search argument, a relative record number. For sequential processing (OPTCD=(KEY,SEQ)), the 4 bytes are required for VSAM to return the feedback RRN. For keyed access (OPTCD=KEY), the search argument is a full or generic key or relative record number. For addressed access (OPTCD=ADR), the search argument is an RBA. If you specify a generic key (OPTCD=GEN), you must also specify in the KEYLEN parameter how many of the bytes of the full key you are using for the generic key. ARG is also used with WRTBFR and MRKBFR. Using WRTBFR and MRKBFR to share resources is described in *z/OS DFSMS Using Data Sets*.

For a VSAMDB base or AIX document in BSON data format, the search argument located at ARG=address must be a full search 'element' in the formal definition of BSON and JSON. For BSON, the argument must be 'type, Key name, Key value length, Key value', such as '024E616D65000B0000004A6F686E20536D69746800' where type=02 (string), key name=4E616D6500 ('Name' in UTF-8), key value length=0B000000 (decimal 11), key value=4A6F686E20536D69746800 ('John Smith').

For JSON, the same search argument would be simply "John Smith", including the "s if it is a string type.

DBARGLN=*abs expression*
specifies the length, in bytes, of the search argument in the field that is specified in ARG=address for a VSAMDB database. Along with ARG, DBARGLN is a required parameter for POINT and GET DIR requests against VSAMDB JSON databases; DBARGLN is optional for BSON. DBARGLN is ignored for non-VSAMDB data sets.

ECB=*address*
specifies the address of an event control block (ECB) you may supply. VSAM indicates in the ECB whether a request is complete or not. For more details see the Event Control Block Fields section in *z/OS DFSMSdfp Advanced Services*. You can use the ECB to determine that an asynchronous request is complete before issuing a CHECK macro. (If you issue a CHECK before a request is complete, you give up control and must wait for completion.) The ECB parameter is always optional.

KEYLEN=*abs expression*
specifies the length, in bytes, of the generic key (OPTCD=GEN) you are using for a search argument (given in the field addressed by the ARG parameter). This parameter is specified as a number from 1 through 255. It is required when the search argument is a generic key. For full-key searches, VSAM knows the key length, which is taken from the catalog definition of the data set when you open the data set. This parameter is ignored for z/OS UNIX files.

MSGAREA=*address*
specifies the address of an area you may, optionally, supply for VSAM to send you a message in case of a physical error. The format of a physical error message is given in "Reason code (physical errors)" on page 138.

MSGLEN=*abs expression*
specifies the size, in bytes, of the message area indicated in the MSGAREA parameter. If MSGAREA is specified, MSGLEN is required. The minimum size of a message is 128 bytes. If you provide less than 128 bytes, no message is returned to your program.

NXTRPL=*address*
specifies the address of the next request parameter list in a chain. Omit this parameter from the macro that generates the last list in the chain. When you issue a request defined by a chain of request parameter lists, indicate in the request macro the address of the first parameter list in the chain. This parameter is not supported for z/OS UNIX files and, if it is specified with a non-zero value results in an error on a subsequent GET, PUT, or POINT.

OPTCD=([ADR|CNV |KEY]
[,DIR|SEQ|SKP]
[,ARD|FRD|LRD]
[,FWD|BWD]
[,ASY|SYN]
[,NSP|NUP|UPD]
[,KEQ|KGE]
[,FKS|GEN]
[,NWAITX|WAITX]
[,LOC|MVE]
[,CR|NRI]
[,ARA31|ARA64]
[,RBA|XRBA])
specifies the subparameters governing the request defined by the request parameter list. Each group of subparameters has a default; subparameters are shown in Table 3 on page 72 with defaults underlined. Only one subparameter from each group can be specified. Some requests do not require a subparameter from all of the groups to be specified. The groups that are not required are ignored. Thus, you can use the same request parameter list for a combination of requests (GET, PUT, POINT, for example) without zeroing out the inapplicable subparameters each time you go from one request to another.

TIMEOUT=*number*
for RLS only, specifies the time in seconds that your program is willing to wait to obtain a lock on a VSAM record when a lock on the record is already held by another program. A non-zero value for TIMEOUT (or if TIMEOUT is not specified) specifies the time (in seconds) this program will wait for the other program(s) to release the lock. A value of zero specifies TIMEOUT processing is *NOT* to be performed by VSAM for this request. That is, if the record lock required by the request is held by another program, the program waits until the other program releases the lock regardless of how long that might be. This parameter is ignored for z/OS UNIX files.

Table 3. OPTCD Options

Option	Meaning
ADR	Addressed access to a key-sequenced or an entry-sequenced data set: RBAs are used as search arguments and sequential access is done by entry sequence. RLS does not support access to a KSDS.
CNV	Control interval access. Control interval access is not allowed for compressed data sets. RLS does not support CNV access. This parameter is ignored for z/OS UNIX files and if it is specified results in an error on a subsequent GET, PUT, or POINT.
KEY	Keyed access to a RRDS or KSDS. Keys or relative record numbers are used as search arguments and sequential access is done by key or relative record number sequence.
DIR	Direct access to a RRDS, KSDS, or ESDS.

Table 3. OPTCD Options (continued)

Option	Meaning
SEQ	Sequential access to a RRDS, KSDS, or ESDS.
SKP	Skip sequential access.
ARD	User's argument determines the record to be located, retrieved, or stored.
LRD	Last record in the data set is to be located (POINT) or retrieved (GET direct); requires OPTCD=BWD.
FRD	For VSAM RLS POINT and GET DIR requests only, FRD, in combination POINT or GET, indicates that the first record or document in the data set or database, respectively, is to be located (by POINT) or retrieved (by GET direct). For regular VSAM RLS data sets, *first* refers to the record with the lowest key or RBA. For VSAMDB databases, *first* refers to the document with the lowest internally generated key, not necessarily the lowest primary key value contained in the documents or the order in which the documents are added. FRD can only be used for POINT or GET DIR. If ARG is specified, ARG is ignored. KEQ, if specified, is also ignored (KEQ is processed as KGE). If SEQ is specified or defaulted for any request, FRD is ignored. FRD and LRD are mutually exclusive. If any request specifies SEQ, FRD is ignored.
FWD	Processing to proceed in a forward direction.
BWD	Processing to proceed in a backward direction; for keyed (KEY) or addressed (ADR) sequential (SEQ) or direct (DIR) requests; valid for POINT, GET, PUT, and ERASE operations; establish positioning by a POINT with OPTCD=BWD or by a GET direct with OPTCD=(NSP,BWD). When OPTCD=BWD is specified, subparameters KGE and GEN are ignored; subparameters KEQ and FKS are assumed. This parameter is ignored for z/OS UNIX files and if it is specified results in an error on a subsequent GET, PUT, or POINT.
ASY	Asynchronous access; VSAM returns to the processing program after scheduling a request so the program can do other processing while the request is being carried out.
SYN	Synchronous access; VSAM returns to the processing program after completing a request.
NSP	With OPTCD=DIR only, VSAM is to remember its position (for subsequent sequential access); that is, the position is not to be forgotten unless an ENDREQ macro is issued.
NUP	A data record being retrieved will not be updated or deleted; a record being stored is a new record; VSAM does not remember its position for direct requests into a work area.
UPD	A data record being retrieved may be updated or deleted; a record being udpated or deleted was previously retrieved with OPTCD=UPD; VSAM remembers its position for sequential and direct GET requests. A GET with update (UPD) must use the same RPL on the following PUT, ERASE or ENDREQ. When PUT, ERASE or ENDREQ is issued after a DIRUPD GET request, VSAM releases exclusive control. This parameter is not supported for z/OS UNIX files, and if it is specified, results in an error on a subsequent GET, PUT, or POINT.

Table 3. OPTCD Options (continued)

Option	Meaning
KEQ	For GET with OPTCD=(KEY,DIR) or (KEY,SKP) and for POINT with OPTCD=KEY, the key (full or generic) that you provide for a search argument must equal the key or relative record number of a record. For a RRDS, KEQ is assumed except for POINT.
KGE	For the same cases as KEQ, if the key (full or generic) that you provide for a search argument does not equal that of a record, the request applies to the record that has the next higher key. If using POINT with a RRDS, KGE positions to the specified relative record number whether the slot is empty or not. If the relative record number is greater than the highest existing record, EOD is returned. A subsequent PUT will insert the record at this position.
FKS	A full key is provided as a search argument.
GEN	A generic key is provided as a search argument; give the length in the KEYLEN parameter. Generic keys are not supported for a variable-length RRDS.
NWAITX	Never take the UPAD or RLSWAIT exit.
WAITX	If OPTCD=SYN and the ACB's MACRF=LSR GSR and UPAD exit routing is specified, VSAM takes the UPAD exit at points when VSAM would normally issue a WAIT. For RLS, take the RLSWAIT exit which is active for this request.
LOC	For retrieval, VSAM leaves the data record in the I/O buffer for processing, unless the data set is compressed, in which case VSAM moves the record to a work area; not valid for PUT or ERASE; valid for GET with OPTCD=UPD. However, to update the record, you must build a new version of the record in a work area and modify the request parameter list OPTCD from LOC to MVE before issuing a PUT. For keyed-sequential retrieval, modifying key fields in the I/O buffer may cause incorrect results for subsequent GET requests until the I/O record is reread. Not valid for requests with spanned records. For z/OS UNIX files, LOC mode is supported but requires extra overhead to get storage in the user space and move the record.
MVE	For retrieval, VSAM moves the data record to a work area for processing, and for storage, VSAM moves it from the work area to the I/O buffer.
CR	For RLS GET and POINT only, CR (consistent read integrity) specifies that a shared lock is to be obtained and released as part of GET processing. CR specifies the application wants this request to be serialized with update/erase of this record by other applications or transactions. RLS obtains a share lock on the record. For RLS POINT, the shared lock remains held on successful completion of the POINT CR request. For RLS GET, after moving a copy of the record to the area pointed to by the RPL AREA parameter, the shared lock is released. If neither NRI, or CR is specified, the NRI/CR option is determined in the following order: • RLSREAD specification on the ACB, if any, • RLS JCL specification, if any, • NRI is assumed. If there are multiple specifications in the RPL, CR takes precedence over NRI.

Table 3. OPTCD Options (continued)

Option	Meaning
NRI	For RLS GET NUP and POINT only, NRI (no read integrity) specifies no locking on a GET(non-update). Since a lock is not obtained on the record, another application or transaction may currently hold an exclusive lock on the record. For a recoverable sphere, the returned record may be an uncommitted change which may be later backed out (this form of processing is sometimes referred to as "dirty read"). The opposite form of read processing is provided by the CR option where if another application/transaction holds an exclusive lock on the record, the reader waits for release of the exclusive lock and thus does NOT read an uncommitted change. If neither NRI or CR is specified, the NRI/CR option is determined in the following order: • RLSREAD specification on the ACB, if any, • RLS JCL specification, if any, • NRI is assumed. If there are multiple specifications in the RPL, CR takes precedence over NRI. Inserting or updating a base cluster record can result in a concurrent NRI read to the record by an alternate index path, causing you to receive a false error (return code 8, reason code 144 in Table 17 on page 124). RLS obtains a record lock and retries the request to be sure this is not a false condition.
ARA31	For VSAM RLS only, ARA31 is the default condition in which RPL AREA and ARG fields contain the address of the VSAM RLS record or VSAMDB (DEFINE CLUSTER with DATABASE specified) document and request argument below the 2 GB bar in the user's address space. ARA31 and ARA64 are mutually exclusive.
ARA64	For VSAM RLS only, allows a user's regular VSAM RLS record or VSAMDB (cluster DEFINEd with DATABASE specified) document to be passed to VSAM RLS above the 2 GB bar in the user's address space. The request's search argument, if any, is also above the 2GB bar. When you specify RPL OPTCD=(ARA64), the user's RPL AREA and ARG fields contain the address of a user-built 8-byte area below the 2 GB bar, which in turn points to the actual record or document and request argument above the bar in the user's address space.
RBA	For addressed accessing (OPTCD=ADR), the ARG field contains the address of a 4-byte RBA. RBA is the default. Extended addressing is not to be used for this request.

Table 3. OPTCD Options (continued)

Option	Meaning
XRBA	For addressed accessing (OPTCD=ADR), the ARG field contains the address of an 8-byte RBA search argument. While you can specify RBA while using XRBA, the following considerations apply to accessing by RBA values: • For a GET extended addressing request, you must specify an OPTCD which includes DIR, ADR, and XRBA. • For a POINT extended addressingrequest, you must specify an OPTCD which includes ADR and XRBA. • For a MRKBFR extended addressingrequest, you must specify an OPTCD which includes XRBA. The ARG field has the address of a 16 byte field containing the beginning and ending 8 byte RBAs of the range. • For a SCHBFR extended addressingrequest, you must specify an OPTCD which includes XRBA. The ARG field has the address of a 16 byte field containing the beginning and ending 8 byte RBAs of the range. • For a WRTBFR TYPE=DRBA extended addressingrequest, you must specify an OPTCD which includes XRBA. The ARG field has the address of an 8 byte field containing the 8 byte RBA to be located and written. If the data being referenced by RBA for an extended addressing KSDS is less than 4GB, you do not have to code this parameter. For data with RBA greater than 4GB the RPL must specify extended addressing (XRBA) and an 8-byte RBA is required. Also, to retrieve an 8-byte RBA using SHOWCB for the RPL, XRBA must be used instead. XRBA specification can be used for any data set (whether or not it is extended addressable).

RECLEN=*abs expression*
specifies the length, in bytes, of a data record being stored. This parameter is required for a PUT request.

For GET requests, VSAM puts the length of the record retrieved in this field in the request parameter list. It will be there if you update and store the record.

TRANSID=*abs expression*
specifies a number that relates modified buffers in a buffer pool. Used in shared resource applications and described in *z/OS DFSMS Using Data Sets*. This parameter is ignored for z/OS UNIX files.

ZHYPERWRITE={YES|NO}
specifies YES to enable zHyperWrite support for the request.

Example: RPL macro

In this example, an RPL macro is used to generate a request parameter list named PARMLIST.

```
ACCESS   ACB   MACRF=(SKP,OUT),                                        x
               DDNAME=PAYROLL

PARMLIST RPL   ACB=ACCESS,                                            x
               AM=VSAM,                                               x
               AREA=WORK,                                             x
               AREALEN=125,                                           x
               ARG=SEARCH,                                            x
               MSGAREA=MESSAGE,                                       x
               MSGLEN=128,                                            x
               OPTCD=(SKP,UPD)   Most OPTCD defaults are appropriate  x
                                 to assumptions.
```

```
WORK     DS    CL125
SEARCH   DS    CL8
MESSAGE  DS    CL128
```

The ACB macro named ACCESS, specifies skip-sequential retrieval for update. Further details may be provided on a DD statement named PAYROLL.

The RPL macro's parameters are:

- ACB associates the request parameter list with the access method control block generated by ACCESS.
- AREA and AREALEN specify a work area, WORK, that is 125 bytes long.
- ARG specifies the search argument is defined at SEARCH. The search argument is 8 bytes long.
- MSGAREA and MSGLEN specify a message area, MESSAGE, that is 128 bytes long. The message area is provided for physical error messages.
- OPTCD specifies skip-sequential processing and specifies a retrieved record may be updated or deleted.
- NSR is assumed.

Because KEYLEN is not coded, a full-key search is assumed.

SCHBFR—Search buffer

If you are using local or global shared resources, you can use the SCHBFR macro to search a buffer.

The format of the SCHBFR macro is:

[*label*]	SCHBFR	[BFRNO=*abs expression*] ,RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the SCHBFR macro.

BFRNO=*abs expression*
specifies the number of the buffer VSAM is to search first. The buffers preceding it in the buffer pool are not searched. The default is 1; that is, the first buffer is searched first. (If the number is coded in register notation, all registers except 1 and 13 may be used.)

The meaning of BFRNO depends on the total number of buffers in the buffer pool and the number of control intervals in the RBA range given by the RPL ARG parameter. This number is the buffer number relative to the beginning of the RBA range if the total number of buffers in the buffer pool is greater than (3/4 x number of CIs in the RBA range)+3. Otherwise, it is the buffer number on the physical buffer chain.

Restriction: When a data set is in a compressed format, records might be compressed and each buffer might contain an unpredictable amount of data.

RPL=*address*
specifies the address of the request parameter list defining the SCHBFR request. These RPL parameters have meaning for SCHBFR:

ACB=*address*
AREA=*address*
If a buffer is found, the area whose address is specified contains its address (OPTCD=LOC) or a copy of its contents (OPTCD=MVE). With compressed data sets, the contents of the buffer will not be in a readable format. SCHBFR is not recommended for compressed data sets.

AREALEN=*abs expression*
At least 4 with OPTCD=LOC; at least control interval size with OPTCD=MVE.

ARG=*address*
ARG gives the address of an 8-byte field containing the beginning and ending control interval RBAs of the range to be searched on. For compressed data sets, the RBA of another record or the

address of the next record in a buffer cannot be determined using the length of the current record or the length of the record provided to VSAM.

For extended addressing, the addressof a 16-byte field containing the beginning and ending 8-byte RBAs of the range.

ECB=*address*

OPTCD=({ASY|SYN},{LOC|MVE})

TRANSID=*abs expression*

All other RPL parameters are ignored. RPLs are assumed not to be chained. Control interval access is assumed.

If the ACB to which the RPL is related has MACRF=GSR, the program issuing SCHBFR must be in supervisor state with protection key 0 to 7.

SHOWCAT—Display the catalog

The information shown here is provided for compatibility only.

The SHOWCAT (show, or display, the catalog) macro enables you to retrieve information from a catalog independently of an open data set defined in the catalog.

The SHOWCAT macro has three forms: standard, list, and execute. Although the integrated catalog facility catalog have different structures, the SHOWCAT macro supports integrated catalog facility catalogs. Thus, all references to catalogs in this discussion of the SHOWCAT macro apply to integrated catalog facility catalogs.

You can use the IGGSHWPL macro to generate a DSECT statement and labels for the fields in the parameter list for SHOWCAT.

The entries in a catalog are interrelated. More than one entry is required to describe an object and its associated objects; one entry points to one or more other entries, which point to yet others. Figure 2 on page 78 shows the interrelationship among entries that describe the following types of objects:

- Alternate index (G)
- Cluster (C)
- Data component (D)
- Index component (I)
- Path (R)
- Upgrade set (Y)

For example, an alternate-index entry points to the entries of its data and index components, its base cluster, and its path. SHOWCAT enables you to follow the arrows in Figure 2 on page 78. You first issue SHOWCAT on the name of an object.

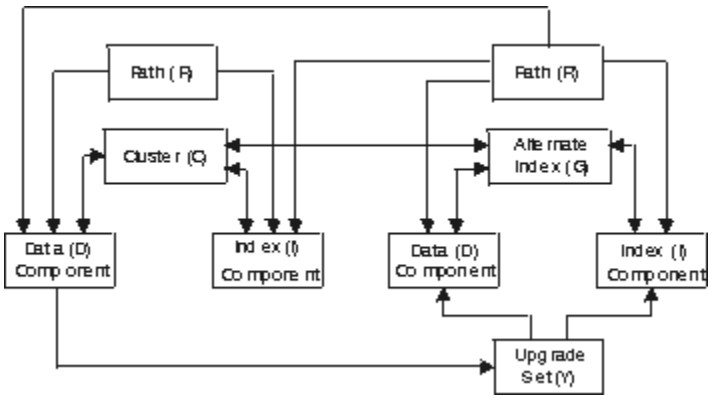


Figure 2. Interrelationship Among Catalog Entries

The information VSAM returns to you includes the control interval numbers of catalog records in entries describing associated objects. You then issue SHOWCAT on a control interval number to retrieve information from one of these other entries.

The first time you issue SHOWCAT, VSAM searches catalogs in the following order to locate the entry that describes the object you name:

1. The master catalog.
2. When the object has a qualified name, the catalog, if any, whose name or alias is the same as the first-level qualifier of the object's name.

VSAM returns the address of the access method control block that defines the catalog. In subsequent use of SHOWCAT, you can specify that address, which causes VSAM to search only that catalog.

SHOWCAT should not be used for z/OS UNIX files as z/OS UNIX files are not reflected in the catalogs. Specifying the pathname in the NAME parameter is not valid and returns unpredictable results.

SHOWCAT is valid in AMODE(24) and AMODE(31). For AMODE(31) callers the address of the parameter list that is passed may be above the line.

SHOWCAT—Standard form

The format of the SHOWCAT macro is:

[*label*]	SHOWCAT	[ACB=*address*] [AREA=*address*] [FWLEN={NO|YES}] [{CI=*address*|NAME=*address*}][RETURN={CI|NAME}]

label
specifies 1 to 8 characters that provide a symbolic address for the SHOWCAT macro.

ACB=*address*
specifies the address of the access method control block that defines the catalog containing the entry from which to display information. You issue the first SHOWCAT without ACB specified and VSAM supplies it to you for the next SHOWCAT (see the description of the work area under the AREA operand). Specifying ACB enables VSAM to go directly to the correct catalog without searching other catalogs first. You should always specify ACB when specifying CI instead of NAME.

AREA=*address*
specifies the address of the work area in which to display the catalog information. The first 2 bytes of the area must give the length of the area, including the 2 bytes. The minimum is 64. If the area is too small, VSAM returns as much information as possible.

FWLEN={NO|YES}
indicates whether the mappings for a workarea returned at the beginning supports a length of a halfword or fullword. The default is **NO**.

NO
indicates that the workarea returned supports only a length of a halfword. This is the default.

YES
indicates that the workarea returned supports only a length of a fullword.

You can use the IGGSHWPL macro to generate a DSECT statement and labels for the fields in the work area.

The format of the work area for RETURN=CI and FWLEN=NO is:

Offset	Length	Symbolic Name	Description
0(X'00')	2	SHWLEN1	Length of the area, including the length of this field (provided by you).
2(X'02')	2	SHWLEN2	Length of the area used by VSAM, including the length of this field and the preceding field.
4(X'04')	4	SHWACBP	The address of the ACB that defines the catalog that contains the entry from which information is displayed.
8(X'08')	1	SHWTYPE	Type of object about which information is returned: **C** Cluster **D** Data component **G** Alternate index **I** Index **R** Path **Y** Upgrade set

Note: For information on the RETURN=NAME and FWLEN=YES workarea mappings see the IGGSHWPL macro.

The following fields contain one set of information for C, G, R, and Y types and another set for D and I types:

The format of the work area for **C, G, R,** and **Y types** is:

Offset	Length or Bit Pattern	Symbolic Name	Description
9(X'09')	1	SHWATTR	For C and Y types: reserved.
			For G type:
	x...	SHWUP	The alternate index may (1) or may not (0) be a member of an upgrade set. One way of verifying this is to display information for the upgrade set of the base cluster and check whether it contains control interval numbers of entries that describe the components of the alternate index. Figure 2 on page 78 shows how to get from the alternate index's catalog entry to the entries that describe its components (G to C to D to Y to D and I).
	.xxx xxxx		Reserved.
			For R type:
	x...	SHWUP	The path is (1) or is not (0) defined for upgrading alternate indexes.
	.xxx xxxx		Reserved.

Offset	Length or Bit Pattern	Symbolic Name	Description
10(X'0A')	2	SHWASS0	The number of association pointers that follow.
		SHWACT	Each association pointer identifies another catalog entry that describes an object associated with this C, G, R, or Y object. The possible types of associated objects are: With C: D, G, I, R. With G: C, D, G, I. With R: C, D, G, I. With Y: D, I. Figure 2 on page 78 shows how the catalog entries for all these objects are interrelated.
12(X'0C')	1	SHWATYPE	Type of object the entry describes.
13(X'0D')	3	SHWAC1	The control interval number of its first record.
16(X'10')			Next association pointer, and so on. For type Y, if the area is too small to display an association pointer for each associated object, VSAM displays as many pointers as possible and returns a code of 4 in register 15. For types C and G, if the area is too small, VSAM displays as many pointers as possible, but returns as a code of 0 in register 15 because fields for the main associated objects can always be displayed (in the smallest allowed work area). For type R, fields for all associated objects (five possible) can always be displayed. (An associated pointer occupies 4 bytes (1 byte for the associated entry type and 3 bytes for its control interval number). However, for all types except Y, 4 additional bytes are required as work space for the SHOWCAT processor. For example, if you provide 80 bytes for associated objects, as many as 10 association pointers can be displayed for type C or G and 20 for type Y.)

The format of the work area for **D** and **I** types is:

Offset	Length	Symbolic Name	Description
9(X'09')	1		Reserved.
10(X'0A')	2	SHWDSB	Relative position of the prime key in records in the data component.
		SHWRKP	For the data component of an ESDS, there is no prime key and this field is 0.
12(X'0C')	2	SHWKEYLN	Length of the prime key.

Offset	Length	Symbolic Name	Description
14(X'0E')	4	SHWCISZ	Control interval size of the data or index component.
18(X'12')	4	SHWMREC	Maximum record size of the data or index component.
22(X'16')	2	SHWASS	The number of association pointers that follow.
		SHWACT	Each association pointer identifies another catalog entry that describes an object associated with this D or I object. The possible types of associated objects are: With D: C, G, Y. With I: C, G. Figure 2 on page 78 shows how the catalog entries for all these objects are interrelated.
24(X'18')	1	SHWATYPE	Type of object the entry describes.
25(X'19')	3	SHWACI	The control interval number of its first record.
28(X'1C')			Next association pointer, and so on. Fields for all associated objects can always be displayed.

{CI=*address*|NAME=*address*}
specifies the address of an area that identifies the catalog entry containing the desired information.

Note: Users of the SHOWCAT macro are strongly urged to convert any uses of the **CI** keyword the **RETURN=NAME** format. The CI implementation exists to be compatible with VSAM catalogs which were no longer usable after 01/01/2000. The CI values returned are not sharable across address spaces and there is a finite table size allowed for mapping component names to pseudo-CI numbers. When this table is exceeded, which can be done by applications that issue the SHOWCAT macro many times, no further information can be returned by the SHOWCAT service. **Because of these limitations the CI interface will be removed in the future.**

CI=*address*
specifies the area is 3 bytes long and contains the control interval number (RBA divided by 512) of the first record in the catalog entry. You can issue the first SHOWCAT with NAME specified, and then VSAM supplies control interval numbers to you for other SHOWCATs (see the description of the work area under the AREA operand). The type of object named must be C, D, G, I, R, or Y. The 3-byte area must be separate from the work area, even though VSAM returns a control interval number in the work area.

NAME=*address*
specifies the area is 44 bytes long and contains the name of the object described by the entry. The name is left-justified and padded with blanks. The type of object named must be C, D, G, I, or R.

RETURN={CI|NAME}
indicates whether mappings will be generated for returning CI numbers or component names. The default is **CI**.

CI
indicates that mappings for CI numbers will be generated and CI numbers will be returned. This is the default.

NAME
indicates that mappings for component names will be generated and component names will be returned.

SHOWCAT—List form

The format of the list form of SHOWCAT is:

[*label*]	SHOWCAT	[ACB=*address*] [AREA=*address*] [FWLEN={NO|YES}] [{CI=*address*|NAME=*address*}] [RETURN={CI|NAME}] MF=L

MF=L
specifies that this is the list form of SHOWCAT.

AREA and {CI|NAME} are optional in the list form of SHOWCAT, but, if they are not so specified, they must be specified in the execute form.

For a detailed description of ACB, AREA, FWLEN, CI|NAME, and RETURN parameters, refer to the information contained in "SHOWCAT—Standard form" on page 79.

SHOWCAT—Execute form

The format of the execute form of SHOWCAT is:

[*label*]	SHOWCAT	[ACB=*address*] [AREA=*address*] [FWLEN={NO|YES}] [{CI=*address*|NAME=*address*}] [RETURN={CI|NAME}] MF=({E|B},*address*)

MF=({E|B},*address*)
specifies this is the execute form of SHOWCAT.

E
specifies the parameter list, whose address is given in *address*, is passed to VSAM for processing.

B
specifies the parameter list is to be built or modified, but is not passed to VSAM. This form of the macro is similar to the list form, except that it works at execution time and can modify a parameter list, as well as build it.

To build a parameter list, first issue SHOWCAT with only MF=(B, address) specified, to zero out the area in which it will be built.

address
specifies the address of the parameter list. If you use register notation, you may use register 1, and a register from 2 through 12. Register 1 is used to pass the parameter list to VSAM (MF=E).

For a detailed description of ACB, AREA, FWLEN, CI|NAME, and RETURN parameters, refer to the information contained in "SHOWCAT—Standard form" on page 79.

Expressions that can be used for SHOWCAT

The values for an operand of SHOWCAT can be expressed as:

- An absolute numeric expression.

- A code or a list of codes separated by commas and enclosed in parentheses.
- A register (in parentheses) from 2 through 12 that contains an address or numeric value. In the execute form of a macro, you can use register 1 for the address of the parameter list. Equated labels can be used to designate a register; for example, BFRNO=(BFR#), where the equate statement, BFR# EQU 3, is included in the program.
- An expression valid for a relocatable A-type address constant; for example, AREA=RETURN+4.

The expressions that can be used depend on the operand. Only absolute numeric expressions, codes, registers, and relocatable A-type address constants are valid for the list form of a macro.

Table 4 on page 84 shows the expressions allowed for each operand of SHOWCAT:

Table 4. Operand Expressions for the SHOWCAT Macro

Operands	Absolute Numeric	Code	Register	A-Type Address
SHOWCAT (STANDARD)				
			X	X
ACB			X	X
AREA			X	X
CI			X	X
NAME				
SHOWCAT (LIST)				
				X
ACB				X
AREA				X
CI		X		
MF				X
NAME				
SHOWCAT (EXECUTE)				
			X	
ACB			X	
AREA			X	
CI				
MF		X		
B		X		
E			X	
address			X	
NAME				

SHOWCB—Display fields of an access method control block

The format of the SHOWCB macro used to display fields in an access method control block is:

[*label*]	SHOWCB	ACB=*address* ,AREA=*address* ,LENGTH=*abs expression* [,OBJECT=DATA|INDEX] ,FIELDS=([ACBLEN] [,AVSPAC] [,BFRFND] [,BSTRNO] [,BUFND] [,BUFNI] [,BUFNO] [,BUFNOL] [,BUFRDS] [,BUFSP] [,BUFUSE] [,CINV] [,CIPCA] [,CDTASIZE] [,DDNAME] [,ENDRBA] [,ERROR] [,EXLST] [,FS] [,HALCRBA] [,HLRBA] [,KEYLEN] [,LEVEL] [,LOKEY] [,LRECL] [,MAREA] [,MLEN] [,NCIS] [,NDELR] [,NEXCP] [,NEXT] [,NINSR] [,NIXL] [,NLOGR] [,NRETR] [,NSSS] [,NUIW] [,NUPDR] [,PASSWD] [,RELEASE] [,RKP] [,SDTASIZE] [,SHRPOOL] [,STMST] [,STRMAX] [,STRNO] [,UIW] [,XAVSPAC] [,XENDRBA] [,XHALCRBA])

The subparameters of the SHOWCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. “Subparameters with GENCB, MODCB, SHOWCB, and TESTCB” on page 5, further defines these operand expressions.

label
specifies 1 to 8 characters that provide a symbolic address for the SHOWCB macro.

ACB=*address*
specifies the address of the access method control block whose fields are displayed. If you used the ACB macro with a label, you can specify the label here. The ACB parameter is optional when you wish to display the length of an access method control block (FIELDS=ACBLEN). (All access method control blocks have the same length, so you need not specify the address of a particular one.)

AREA=*address*
specifies the address of a return area you are supplying for VSAM to display the contents of the fields specified in the FIELDS parameter. The contents of the fields are displayed in the order in which you specify them. The area must begin on a fullword boundary.

LENGTH=*abs expression*
specifies the length, in bytes, of the return area you are providing for VSAM to display the indicated fields in. (See the FIELDS parameter for the fields that can be displayed and for the length of each field.) If the area is not large enough for all the fields, VSAM does not display any of their contents and returns a reason code (see “Control block manipulation macro return and reason codes” on page 118).

OBJECT=DATA|INDEX
specifies whether fields are displayed for the data or for the index.

FIELDS=[ACBLEN][,AVSPAC]
[,BFRFND][,BSTRNO]
[,BUFND][,BUFNI]
[,BUFNO][,BUFNOL][,BUFRDS]
[,BUFSP][,BUFUSE]
[,CDTASIZE][,CINV][,CIPCA]
[,DDNAME][,ENDRBA]
[,ERROR][,EXLST]
[,FS][,HALCRBA][,HLRBA]
[,KEYLEN][,LEVEL]
[,LOKEY][,LRECL]
[,MAREA][,MLEN]
[,NCIS][,NDELR]
[,NEXCP][,NEXT]
[,NINSR][,NIXL]
[,NLOGR][,NRETR]
[,NSSS][,NUIW]
[,NUPDR][,PASSWD]
[,RELEASE][,RKP][,SHRPOOL]
[,STMST][,STRMAX]
[,SDTASIZE]
[,STRNO][,UIW]
[,XAVSPAC][,XENDRBA]
[,XHALCRBA])

specifies the fields whose contents are to be displayed. Some of the fields can be displayed at any time; others only after a data set is opened. The ones that can be displayed only after a data set is opened can, for a KSDS that has been opened for keyed access, pertain either to the data or to the index. See the OBJECT parameter.

Table 5 on page 86 explains the subparameters you can code in the FIELDS parameter for an access method control block.

Table 5. FIELDS Keyword Subparameters for an Access Method Control Block

Subparameter	Fullwords	Description of the Field
		Note: The following fields can be displayed at any time.
ACBLEN	1	Length of an access method control block (displaying the length of an access method control block gives your program independence from changes in the length that may occur from release to release of VSAM).
BSTRNO	1	Number of strings initially allocated for access to the base cluster by a path. For RLS BSTRNO is ignored and the value specified in the ACB is returned.
BUFND	1	Number of I/O buffers used for data, as specified in the ACB (or GENCB). For RLS BUFND is ignored and the value specified in the ACB is returned. This parameter has no effect for z/OS UNIX files.
BUFNI	1	Number of I/O buffers used for index entries, as specified in the ACB (or GENCB). For RLS BUFNI is ignored and the value specified in the ACB is returned. This parameter has no effect for z/OS UNIX files.
BUFSP	1	Amount of space specified in the ACB (or GENCB) for I/O buffers. For RLS BUFSP is ignored and the value specified in the ACB is returned. This parameter has no effect for z/OS UNIX files.
DDNAME	2	Name of the DD statement that identifies the data set.

Table 5. FIELDS Keyword Subparameters for an Access Method Control Block (continued)

Subparameter	Fullwords	Description of the Field
ERROR	1	The code returned by VSAM after the opening or closing of the data set (see "OPEN—Connect program and data" on page 59 and "CLOSE—Disconnect program and data" on page 24).
EXLST	1	Address of the exit list, if any; 0 if none.
LEVEL	2	Address (in first fullword) and length (in second fullword) of the field containing the DFP level information.
MAREA	1	Address of the message area, if any; 0 if none.
MLEN	1	Length of the message area, if any; 0 if none.
PASSWD	1	Address of the field containing the password; the first byte of the field contains the length of the password (in binary). This parameter has no effect for z/OS UNIX files.
RELEASE	2	Address (in first fullword) and length (in second fullword) of the field containing the DFP release information. *z/OS DFSMSdfp Advanced Services* discusses how to use the IHADFA mapping macro or the IGWASYS callable service for release determination.
SHRPOOL	1	Identification number of resource pool to be used for LSR processing. SHRPOOL specification is ignored by RLS processing. This parameter has no effect for z/OS UNIX files.
STRMAX	1	Maximum number of strings concurrently active. For RLS this field is the number of active strings associated with this ACB at the time the request is issued.
STRNO	1	Number of requests for which VSAM is prepared to remember its position in the data set. For RLS the value specified in the ACB macro is ignored. After OPEN a value of 1024 is returned, indicating the maximum number of strings allowed.
		Rule: The following fields can be displayed only after the data set is opened. It is your responsibility to be sure that the ACB remains open until the SHOWCB for these fields has completed. If the ACB is closed while a SHOWCB is active for these fields, unpredictable results can occur including abends.
AVSPAC	1	Amount of available space in the data component or index component, in bytes. If the extended format data set might contain more than 4GB, use XAVSPAC instead of AVSPAC.
BFRFND	1	Number of successful look-asides. For RLS this field is the number of requests satisfied from the local cache or the CF cache.
BUFNO	1	Number of I/O buffers allocated for the data component or index component. The value of zero will be returned for GSR, LSR, RLS and UNIX System services.
BUFNOL	1	Number of I/O buffers allocated for the data component or index component during BLDVRP or SMB for LSR processing. The value of zero will be returned if the ACB is not specified for LSR processing.
BUFRDS	1	Number of buffer reads. For RLS this field is the number of times I/O is done for a READ.

Table 5. FIELDS Keyword Subparameters for an Access Method Control Block (continued)

Subparameter	Fullwords	Description of the Field
BUFUSE	1	Number of I/O buffers actually in use for the data component or index component at the time the SHOWCB macro issued. The value of zero will be returned if the ACB is not specified for NSR or LSR processing.
CDTASIZE	2	Value for the size of extended format data sets using compression. For RLS, this number is maintained on an ACB basis rather than a component basis. That is, it is a count only for the current open of the data set with this ACB. It is zero if no record has been processed since the current open.
CINV	1	Control interval size for the data component or index component.
CIPCA	1	Number of control intervals for each control area.
ENDRBA	1	Ending RBA of the space used by the data component or index component; not the RBA of any record in the data set, but of the last used byte in the data set (high-used RBA). If the extended format data set might contain more than 4GB, use XENDRBA instead of ENDRBA.
FS	1	Number of free control intervals per control area in the data component (0 for OBJECT=INDEX). For z/OS UNIX files this field is set to zero.
HALCRBA	1	High-allocated RBA; the relative byte address of the end of the data component (OBJECT=DATA) or the index component (OBJECT=INDEX). If the extended format data set might contain more than 4GB, use XHALCRBA instead of HALCRBA.
HLRBA	1	RBA of the highest-level index control interval.
KEYLEN	1	Length of the key of reference of the key field of data records in the data component (whether OBJECT=DATA or INDEX).
LOKEY	2	Address of the field containing the low key (in first fullword) and the length (in second fullword) of the low key of a KSDS data component. For RLS LOKEY is not supported. A reason code is given if it is specified.
LRECL	1	Length of data records in the data component (maximum length for variable-length data records) or of index records in the index component (control interval length minus 7).
NCIS	1	Number of control intervals split in the data component (0 for OBJECT=INDEX). For z/OS UNIX files this field is set to zero. For RLS, this number is maintained on an ACB basis rather than a component basis. That is, it is a count only for the current open of the data set with this ACB. It is zero if no record has been processed since the current open.
NDELR	1	Number of records deleted from the data component (0 for OBJECT=INDEX). For RLS, this number is maintained on an ACB basis rather than a component basis. That is, it is a count only for the current open of the data set with this ACB. It is zero if no record has been processed since the current open.

Table 5. FIELDS Keyword Subparameters for an Access Method Control Block (continued)

Subparameter	Fullwords	Description of the Field
NEXCP	1	Number of I/O requests VSAM has issued for access to the data component or index component. For RLS NEXCP is a count of the number of calls to the system buffer manager (includes calls that result in either a CF cache access or an I/O).
NEXT	1	Number of extents now allocated to the data component or index component (the maximum that can be allocated is 123 per VSAM component. For z/OS UNIX files this field is set to one.
NINSR	1	Number of records inserted into (or added to) the data component (0 for OBJECT=INDEX). For RLS, this number is maintained on an ACB basis rather than a component basis. That is, it is a count only for the current open of the data set with this ACB. It is zero if no record has been processed since the current open.
NIXL	1	Number of levels in the index component (0 for OBJECT=DATA).
NLOGR	1	Number of records in the data component or index component.For z/OS UNIX files this field is set to zero.
NRETR	1	Number of records that have ever been retrieved from the data component (0 for OBJECT=INDEX). For RLS, this number is maintained on an ACB basis rather than a component basis. That is, it is a count only for the current open of the data set with this ACB. It is zero if no record has been processed since the current open.
NSSS	1	Number of control areas split in the data component (0 for OBJECT=INDEX). For z/OS UNIX files this field is set to zero. For RLS, this number is maintained on an ACB basis rather than a component basis. That is, it is a count only for the current open of the data set with this ACB. It is zero if no record has been processed since the current open.
NUIW	1	Number of writes not initiated by the user. For RLS NUIW does not apply, and is set to zero.
NUPDR	1	Number of updated records in the data component or index component. For RLS, this number is maintained on an ACB basis rather than a component basis. That is, it is a count only for the current open of the data set with this ACB. It is zero if no record has been processed since the current open.
RKP	1	Displacement of the key of reference of the key field from the beginning of a data record (whether OBJECT=DATA or INDEX).
RMODE31	1	For VSAM, with or without SMB (system-managed buffering), the effective RMODE31 setting: 0 if NONE, 1 if BUFF, 2 if CB, and 3 if ALL For RLS, RMODE31 is invalid and the field will be ignored during Open.

Table 5. FIELDS Keyword Subparameters for an Access Method Control Block (continued)

Subparameter	Fullwords	Description of the Field
SDTASIZE	2	Value for the amount of source data for extended format data sets using compression. For RLS, this number is maintained on an ACB basis rather than a component basis. That is, it is a count only for the current open of the data set with this ACB. It is zero if no record has been processed since the current open.
STMST	2	System time stamp, which gives the time and day of the last time the data component or index component was closed, with bit 51 (counting from 0 at the left) equivalent to one microsecond and bits 52 through 63 unused. For z/OS UNIX files this field is set to the time of day of the current open.
UIW	1	Number of user-initiated writes. For RLS UIW does not apply, and is set to zero.
XAVSPAC	2	Amount of available space in the data component or index component, in bytes. XAVSPAC (instead of AVSPAC) specifies the return area (you are providing for VSAM for display) is two full words long to contain values possibly greater than 4GB.
XENDRBA	2	Ending RBA of the space used by the data component or index component; not the RBA of any record in the data set, but of the last used byte in the data set (high-used RBA). XENDRBA (instead of ENDRBA) specifies the return area (you are providing for VSAM for display) is two full words long to contain values possibly greater than 4GB.
XHALCRBA	2	High-allocated RBA; the relative byte address of the end of the data component (OBJECT=DATA) or the index component (OBJECT=INDEX). XHALCRBA (instead of HALCRBA) specifies the return area (you are providing for VSAM for display) is two full words long to contain values possibly greater than 4GB.

Example 1: SHOWCB macro (display an access method control block)

In this example, a SHOWCB macro is used to display fields in an access method control block. The fields displayed (KEYLEN, LRECL, and RKP) permit the program to modify variables to process any one of many data sets that have different sized key fields and records and different placements of key field in a record.

```
         SHOWCB ACB=CONTROL,                                           x
                AREA=DISPLAY,                                          x
                FIELDS=(KEYLEN,                                        x
                LRECL,RKP),                                            x
                LENGTH=12

DISPLAY  DS     0F              Align on fullword boundary.
KEYLEN   DS     F
LRECL    DS     F
RKP      DS     F
```

The SHOWCB macro's parameters are:

- ACB specifies the address of the access method control block to be displayed.

- AREA specifies the area used to display access method control block fields begins on a fullword boundary.
- FIELDS specifies the KEYLEN, LRECL, and RKP fields are displayed.
- LENGTH specifies the length of the area used for the display is 12 bytes, enough to accommodate the specified fields.

This display allows the program to set up its variables for the particular data set it has opened.

Example 2: SHOWCB macro (display an exit list address)

In this example, a SHOWCB macro is used to get the address of an exit list by displaying the address in an access method control block that uses the exit list.

```
        SHOWCB ACB=address,                                          x
               AREA=address,                                         x
               FIELDS=EXLST,                                         x
               LENGTH=4
```

The SHOWCB macro's parameters are:

- ACB specifies the address of an access method control block from which the address of an exit list is displayed.
- AREA and LENGTH specify an area and length, 4 bytes, used to display the address of the exit list.
- FIELDS specifies the EXLST field in an access method control block is displayed.

Important: If you issue a SHOWCB for a non-VSAM and non-VTAM ACB, the results will be unpredictable.

SHOWCB—Display fields of an exit list

The format of the SHOWCB macro used to display fields in an exit list is:

[*label*]	SHOWCB	EXLST=*address* ,AREA=*address* ,LENGTH=*abs expression* ,FIELDS=([EODAD] [,EXLLEN] [,JRNAD] [,LERAD][,SYNAD])

The subparameters of the SHOWCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. “Subparameters with GENCB, MODCB, SHOWCB, and TESTCB” on page 5, further defines these operand expressions.

label
specifies 1 to 8 characters that provide a symbolic address for the SHOWCB macro.

EXLST=*address*
specifies the address of the exit list whose fields are displayed. If you used the EXLST macro with a label, you can specify the label here. The EXLST parameter is optional only when you want to display the length an exit list can have (see FIELDS=EXLLEN below). The SHOWCB macro does not support the UPAD user exit.

AREA=*address*
specifies the address of a return area you supply for VSAM to display the contents of the fields specified in the FIELDS parameter. The contents of the fields are displayed in the order specified. The area must begin on a fullword boundary.

LENGTH=*abs expression*
specifies the length, in bytes, of the return area you provide for VSAM to display the indicated fields in. Each exit-list field requires a fullword. If the area is not large enough for all the fields, VSAM does not display any of their contents and returns an error code (see “Control block manipulation macro return and reason codes” on page 118).

**FIELDS=([EODAD][,EXLLEN][,JRNAD]
[,LERAD][,SYNAD])**
specifies the values to display, as follows:

EODAD
specifies the address of the end-of-data-set routine is displayed.

EXLLEN
specifies the length of the exit list indicated in the EXLST parameter or if EXLST is omitted, the maximum length an exit length can have, is displayed.

JRNAD
specifies the address of the journalizing routine is displayed.

LERAD
specifies the address of the logical error analysis routine is displayed.

SYNAD
specifies the address of the physical error analysis routine is displayed.

You can use SHOWCB to display the address of an exit routine only if the exit routine is indicated in the exit list. If it is not, the SHOWCB request fails. Use TESTCB to test whether an entry for a given exit type is present in the exit list and to find out whether the exit is active and the routine is to be loaded.

Example: SHOWCB macro (display the length of an exit list)

In this example, a SHOWCB macro is used to display the maximum length of an exit list. The maximum length of an exit list is subsequently used in a GENCB macro to get virtual storage for an exit list.

```
        SHOWCB  AREA=LENGTH,                                          x
                FIELDS=EXLLEN,                                        x
                LENGTH=4

        L       0,LENGTH          Amount of storage for GETMAIN.
        GETMAIN R,LV=(0)
        LR      2,1               Address of storage for GENCB.

        GENCB   BLK=EXLST,        Indirect notation for length of return x
                LENGTH=(*,        area.                               x
                LENGTH),                                              x
        .       WAREA=(2)
        .
LENGTH  DS      F                 Contains the length of GENCB's return x
                                  area.
```

The SHOWCB macro's parameters are:

- AREA and LENGTH specify the area, which begins on a fullword boundary, and its length, 4 bytes, that is used for the display.
- FIELDS specifies that the maximum length of an exit list is displayed. Because only EXLLEN is specified, the EXLST parameter is omitted.

The GENCB macro specifies a return area in which an exit list is to be generated. The length of the return area is located at LENGTH, where the maximum length of an exit list was put as a result of the SHOWCB macro.

SHOWCB—Display fields of a request parameter list

The format of the SHOWCB macro used to display fields in a request parameter list is:

[*label*]	SHOWCB	RPL=*address* ,AREA=*address* ,LENGTH=*abs expression* ,FIELDS=([ACB][,AIXPC][,AREA][,AREALEN] [,ARG][,DBARGLN][,ECB][,FDBK][,FTNCD] [,KEYLEN][,MSGAREA] [,MSGLEN] [,NXTRPL][,RBA] [,RECLEN] [,RPLLEN] [,TRANSID] [,XRBA])

The subparameters of the SHOWCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Subparameters with GENCB, MODCB, SHOWCB, and TESTCB" on page 5, further defines these operand expressions.

label
specifies 1 to 8 characters that provide a symbolic address for the SHOWCB macro.

RPL=*address*
specifies the address of the request parameter list whose fields are displayed. If you used the RPL macro with a label, you can specify the label here. The RPL parameter is optional when you want to display the length of a request parameter list (FIELDS=RPLLEN). (All VSAM request parameter lists have the same length, so you need not specify the address of a particular one.)

AREA=*address*
specifies the address of a return area you supply for VSAM to display the contents of the fields specified in the FIELDS parameter. The contents of the fields are displayed in the order specified. The area must begin on a fullword boundary.

LENGTH=*abs expression*
specifies the length, in bytes, of the return area you provide for VSAM to display the indicated fields in. Each request parameter list field requires a fullword. If the area is not large enough for all the fields, VSAM does not display any of their contents and returns an error code (see "Control block manipulation macro return and reason codes" on page 118).

FIELDS=([ACB][,AIXPC][,AREA][,AREALEN][,ARG]
[,DBARGLN][,ECB][,FDBK][,FTNCD][,KEYLEN]
[,MSGAREA][,MSGLEN]
[,NXTRPL][,RBA][,RECLEN]
[,RPLLEN][,TRANSID]
[,XRBA][,TRANSID])
specifies the fields whose contents are displayed. Table 6 on page 93 explains the subparameters you can code in the FIELDS parameter for a request parameter list.

Table 6. FIELDS Keyword Subparameters for a Display Request Parameter List

Subparameter	Fullwords	Description of the Field
ACB	1	Address of the access method control block that relates the request parameter list to the data.
AIXPC "1" on page 94	1	Number of alternate index pointers.
AREA	1	Address of the return area the program uses to process a data record for the access as defined by the request parameter list.
AREALEN	1	Length of the return area whose address is given in AREA.

Table 6. FIELDS Keyword Subparameters for a Display Request Parameter List (continued)

Subparameter	Fullwords	Description of the Field
ARG	1	Address of the field containing a search argument, if search arguments are being used.
DBARGLN	1	Length of the search argument in the field that is specified in RPL ARG for a VSAMDB database. Along with ARG, DBARGLN is required for POINT and GET requests against JSON databases; DBARGLN is optional for BSON. DBARGLN is ignored for non-VSAMDB data sets.
ECB "1" on page 94	1	Address of an event control block, if any, in which VSAM indicates the completion of requests defined by the request parameter list.
FDBK "1" on page 94	1	Reason code that VSAM puts into the feedback field to describe the error detected for the preceding request. (The meaning of this code depends on the contents of register 15, which indicates whether the request was successful or failed because of a logical or physical error. See "Record management return and reason codes" on page 120.)
FTNCD "1" on page 94	1	Code that describes the function in which a logical or physical error occurred; indicates whether the upgrade set may have been modified incorrectly by the preceding request. (The meaning of this code depends on the contents of register 15, which indicates whether the request was successful or failed because of a logical or physical error. See "Record management return and reason codes" on page 120.)
KEYLEN	1	Length of the search argument, if a generic key is used for a search argument.
MSGAREA "1" on page 94	1	Address of the area, if any, into which VSAM puts physical error messages.
MSGLEN	1	Length of the message area, if any.
NXTRPL	1	Address of the next request parameter list, if another one is chained to this one.
RBA "1" on page 94	1	Relative byte address of the most recently processed record; you could use it to record the RBAs of records that you are retrieving or storing sequentially or by key.
RECLEN "1" on page 94	1	Length of the data record, access to which is defined by the request parameter list.
RPLLEN	1	Length of a request parameter list.
TRANSID	1	Number that relates modified buffers in a buffer pool; described in *z/OS DFSMS Using Data Sets*.
XRBA "1" on page 94	2	The return area (you are providing for VSAM for display) is two full words long to contain values possibly greater than 4GB.

Note:

1. These fields are significant only if the requests are completed. Therefore, you must wait until the request completes (for example, by issuing a CHECK if the request is asynchronous) before issuing SHOWCB.

Example: SHOWCB macro (display a physical error message)

In this example, a SHOWCB macro is used to display a physical error message. This example assumes that there is no SYNAD routine (or the SYNAD exit is inactive). In this case, VSAM returns control to

your program following the last executable instruction if a physical error occurs. Register 15 indicates a physical error (12), and the feedback field in the request parameter list contains a code identifying the error. The message area contains more details about the error. Register 1 points to the request parameter list.

```
REQUEST RPL    MSGAREA=MESSGES, MSGLEN=128                              x
        .
        SHOWCB AREA=MSGADDR,                                            x
               FIELDS=MSGAREA,                                          x
               LENGTH=4,                                                x
               RPL=REQUEST

        LTR    15,15
        BNZ    CHECKO
        .
CHECKO  ...                         Display failed.
        .
```

```
MESSGES DS     CL128                 For VSAM to give you a detailed   x
                                     message about a physical error.

MSGADDR DS     F                     For displaying the address of the x
                                     message area with SHOWCB.
```

The RPL macro in this example provides for a message area, MESSGES, of 128 bytes to be used for any physical error message.

The SHOWCB macro's parameters are:

- AREA and LENGTH specify a 4-byte area, MSGADDR, used for displaying the address of the message area for the associated request parameter list.
- FIELDS specifies the address of the message area is displayed.
- RPL specifies the name, REQUEST, of the request parameter list for which the message area address is displayed.

SHOWCB—List form

The format of the list form of SHOWCB is:

[*label*]	SHOWCB	[{ACB|EXLST|RPL}=*address*] ,AREA=*address* ,FIELDS=(*keyword*[,*keyword*,...]) ,LENGTH=*abs expression* ,MF={L|(L,*address*[,*label*])} ,[OBJECT={DATA|INDEX}]

SHOWCB—Execute form

The format of the execute form of SHOWCB is:

[*label*]	SHOWCB	[{ACB|EXLST|RPL}=*address* ,AREA=*address* ,MF=(E,*address*) [,OBJECT={DATA|INDEX}]

SHOWCB—Generate form

The format of the generate form of SHOWCB is:

[*label*]	SHOWCB	[{ACB|EXLST|RPL}=*address*] ,AREA=*address* ,FIELDS=(*keyword*[,*keyword*,...]) ,LENGTH=*number* ,MF=(G,*address*[,*label*]) [,OBJECT={DATA|INDEX}]

TESTCB—Test a field of an access method control block

Only one keyword can be specified each time you issue TESTCB.

The format of the TESTCB macro used to test a field in an access method control block is:

[*label*]	TESTCB	ACB=*address* [,ERET=*address*] [,OBJECT=DATA|INDEX] ,{ATRB=([ESDS][,KSDS][,LDS][,REPL] [,RRDS][,SPAN][,SSWD][,VRRDS][,WCK])| ATRB=COMPRESS| ATRB=UNQ| ATRB=XADDR| MACRF=([ADR][,AIX][,CFX][,CNV][,DDN] [,DFR][,DIR][,DSN][,GSR][,ICI][,IN] [,KEY][,LEW][,LSR][,NCI][,NDF][,NFX][,NIS] [,NLW][,NRM][NRS][,NSR][,NUB][,OUT][,RLS][,RST] [,SEQ][,SIS][,SKP][,UBF])| OFLAGS=OPEN| OPENOBJ={PATH|BASE|AIX}| ACBLEN=*abs expression*| AVSPAC=*abs expression*| BSTRNO=*abs expression*| BUFND=*abs expression*| BUFNI=*abs expression*| BUFNO=*abs expression*| BUFSP=*abs expression*| CINV=*abs expression*| DDNAME=*character string*| ENDRBA=*abs expression*| ERROR=*abs expression*| EXLST=*address*| FS=*abs expression*| KEYLEN=*abs expression*| LRECL=*abs expression*| MAREA=*address*| MLEN=*abs expression*| NCIS=*abs expression*| NDELR=*abs expression*| NEXCP=*abs expression*| NEXT=*abs expression*| NINSR=*abs expression*| NIXL=*abs expression*| NLOGR=*abs expression*| NRETR=*abs expression*| NSSS=*abs expression*| NUPDR=*abs expression*| PASSWD=*address*| RKP=*abs expression*| SHRPOOL=*abs expression*| STMST=*address*| STRNO=*abs expression*}

The subparameters of the TESTCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. “Subparameters with GENCB, MODCB, SHOWCB, and TESTCB” on page 5, further defines these operand expressions.

ACB=*address*
: specifies the address of the access method control block whose information you want to test. Omit it only if you are testing the length of an access method control block (ACBLEN=number). (All VSAM access method control blocks have the same length.)

ERET=*address*
specifies the address of a routine to which VSAM gives control if an error occurs and VSAM is unable to test for the specified condition. For example, testing AVSPAC in an access method control block for an unopened data set would fail. VSAM indicates in register 15 whether it could do the test and, if not, indicates in register 0 the reason it could not. (The reasons are discussed under "Control block manipulation macro return and reason codes" on page 118.) A failure trying to execute TESTCB indicates a basic logical problem in the processing program, so the error routine would probably issue an ABEND. If it lets the program continue, it must branch to the continuation point itself, and not return to VSAM.

OBJECT={DATA|INDEX}
specifies whether to test a field for data or for index.

ATRB=([ESDS][,KSDS][,LDS]
[,REPL]
[,RRDS]
[,SPAN]
[,SSWD]
[,VRRDS]
[,WCK])
specifies, for an open data set, the attribute to be tested for, as follows:

ESDS
specifies entry-sequenced data set.

KSDS
specifies key-sequenced data set.

LDS
specifies linear data set.

When specified, LDS must be the only parameter indicated by ATRB. All other parameters are ignored and a binary test performed indicating whether the data set is a linear data set (return code 0) or not (return code 1).

REPL
specifies that some portion of the index is replicated.

RRDS
specifies relative record data set.

SPAN
specifies that the data set contains spanned records.

SSWD
specifies that the sequence set is adjacent to the data.

VRRDS
specifies variable-length relative record data set.

WCK
specifies that the write operations for the data set are being verified.

ATRB=COMPRESS
specifies if the data set is in compressed format.

ATRB=UNQ
specifies, for an open alternate index or path, that the alternate index requires unique keys. The test for ATRB=UNQ must be made with a separate TESTCB macro. VSAM examines the path control blocks for the UNQ attribute. VSAM also examines the base cluster's control blocks for the other attributes. If other attributes are tested for, VSAM examines the base cluster's control blocks for all attributes. The test for ATRB=UNQ would give inaccurate results when applied to the base cluster's control blocks.

ATRB=XADDR
specifies if the data set is in extended addressability format.

MACRF=([ADR][,AIX][,CFX]
[,CNV] [,DDN]
[,DFR] [,DIR]
[,DSN] [,GSR]
[,ICI][,IN]
[,KEY][,LEW]
[,LSR][,NCI]
[,NDF][,NFX]
[,NIS][,NLW]
[,NRM][,NRS]
[,NSR][,NUB]
[,OUT][,RLS][,RST]
[,SEQ][,SIS]
[,SKP][,UBF]
specifies that a test is made to determine, at any time, what subparameter or combination of subparameters is being used for processing.

OFLAGS=OPEN
specifies that a test is made to determine, after open, whether the data set identified by the control block was opened.

OPENOBJ=PATH|BASE|AIX
specifies that a test is made to determine, after open, whether an opened object is a path, a base cluster, or an alternate index.

Note: When OPENOBJ is used with an ACB opened for an alternate index, both OPENOBJ=AIX and OPENOBJ=BASE return TRUE (PSW condition code = 0). When OPENOBJ is used with an ACB opened for a path, only OPENOBJ=PATH returns TRUE.

The remaining parameters represent fields in an access method control block that can be compared with the value specified. These fields are the same as those that can be displayed by using the SHOWCB macro and are described in Table 5 on page 86.

If you omit a routine to handle error conditions, you can examine register 15 following TESTCB by using a branch table, for example, but do not alter the PSW condition code that VSAM set to indicate the result of a test until you have tested it.

Important: If you issue a TESTCB for a non-VSAM and non-VTAM ACB, the results will be unpredictable.

Example: TESTCB macro (test for data set attributes)

In this example, a TESTCB macro is used to determine whether a data set is a key sequenced or an ESDS.

```
LIST    RPL
        .
        SHOWCB AREA=DATAFCT,                                          x
               FIELDS=ACB,                                            x
               LENGTH=4,                                              x
               RPL=LIST

        LTR    15,15
        BNZ    CHECKO

        TESTCB ACB=(*,           Is the data set key sequenced?       x
               DATAFCT),                                              x
               ATRB=KSDS,                                             x
               ERET=CHECKO

        BE     KEYSEQ            Yes.
        .
KEYSEQ  ...                      Data set is key sequenced.

CHECKO  ...                      Display or test failed.
        .
DATAFCT DS     F                 For displaying address of access     x
                                 method control block.
```

The SHOWCB macro's parameters are:

- AREA and LENGTH specify a 4-byte area, DATAFCT, aligned on a fullword boundary, used for the display.
- FIELDS and RPL specify the address of the access method control block in the LIST request parameter list to be displayed.

The TESTCB macro's parameters are:

- ACB specifies that a field in the access method control block, the address of which is located at DATAFCT, is to be tested. The SHOWCB macro put the address of the access method control block at DATAFCT.
- ATRB specifies that the access method control block is to be tested to determine whether it is a KSDS.
- ERET specifies that a routine named CHECK0 is to be given control if an error occurs that makes it impossible to make the test.

There is no need to examine the feedback field in an EODAD routine. It can be assumed to contain the end-of-data-set indication.

TESTCB—Test a field of an exit list

The format of the TESTCB macro used to test fields in an exit list is:

[*label*]	TESTCB	EXLST=*address* [,ERET=*address*] ,{EODAD={0|([*address*][,A|N][,L])}| JRNAD={0|([*address*][,A|N][,L])}| LERAD={0|([*address*][,A|N][,L])}| SYNAD={0|([*address*][,A|N][,L])}} [,EXLLEN=*abs expression*]

The subparameters of the TESTCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Subparameters with GENCB, MODCB, SHOWCB, and TESTCB" on page 5, further defines these operand expressions.

label
specifies 1 to 8 characters that provide a symbolic address for the TESTCB macro.

EXLST=*address*
specifies the address of the exit list whose information you want to test. You may omit it only if you are testing the maximum length of an exit list (EXLLEN=number). The TESTCB macro does not support the UPAD user exit.

ERET=*address*
specifies the address of a routine to which VSAM gives control if an error occurs and VSAM is unable to test for the specified condition. For example, testing AVSPAC in an access method control block for an unopened data set would fail. VSAM indicates in register 15 whether it could do the test and, if not, indicates in register 0 the reason it could not. (The reasons are discussed under "Control block manipulation macro return and reason codes" on page 118.) A failure trying to execute TESTCB indicates a basic logical problem in the processing program, so the error routine would probably issue an ABEND. If it lets the program continue, it must branch to the continuation point itself, and not return to VSAM.

EODAD={0|([*address*][,A|N][,L])}|
JRNAD={0|([*address*][,A|N][,L])}|
LERAD={0|([*address*][,A|N][,L])}|
SYNAD={0|([*address*][,A|N][,L])}
specifies the exit about which you are asking a yes-no question. If you code more than one parameter for an exit name, each must equal the corresponding value in the control block for you to get an equal condition. The values that can be tested are:

0
specifies that a test is to be made to determine whether an entry is provided for the exit in the exit list.

address
specifies that a test is to be made to determine whether this is the address of the exit. Tests for an address result in an equal, unequal, high, low, not-high, or not-low condition. Tests for a combination of an address and A, N, or L result in an equal or unequal condition.

A|N
specifies that a test is to be made to determine whether an exit is active (A) or not active (N). Tests for A or N result in an equal or unequal condition.

L
specifies that a test is to be made to determine whether the address is the location of an 8-byte field containing the name of a module to be loaded rather than the entry point of the routine. Tests for L result in either an equal or unequal condition.

EXLLEN=*abs expression*
specifies either the maximum length that an exit list can have (if you do not code the EXLST parameter) or the actual length of the exit list that is indicated by the EXLST parameter. If you specify an exit, you may not also specify EXLLEN. If you specify EXLLEN, you may not also specify an exit.

If you omit a routine to handle error conditions, you can examine register 15 following TESTCB by using a branch table. Do not alter the PSW condition code that VSAM set to indicate the result of a test until you have tested it.

Example: TESTCB macro (use a branch table)

In this example, a TESTCB macro is used to test whether ENDPROC is the routine supplied for the EODAD exit in the exit list EXITS, and whether the EODAD exit is active. A branch table is used to determine whether the test is successful.

```
         TESTCB EODAD=(ENDPROC,A), Is ENDPROC supplied and is the exit  x
               EXLST=EXITS         active?
         B      *+4(15)
```

If the test was made successfully, register 15 contains 0 and the next instruction is executed.

```
         B      TEST1
```

If it was unsuccessful, register 15 contains 4 and the next instruction is executed.

```
         ABEND  2,DUMP

TEST1    BNE    NO

YES      ...                        Yes; ENDPROC is supplied and active.

NO       ...                        ENDPROC is not supplied, or the exit
                                    is not active.
```

TESTCB—Test a field of a request parameter list

The format of the TESTCB macro to test fields in a request parameter list is:

[*label*]	TESTCB	RPL=*address* [,ERET=*address*] ,{AIXFLAG=AIXPKP| AIXPC=*abs expression*| FTNCD=*abs expression*| IO=COMPLETE| OPTCD=([ADR][,ARD][,ASY][,BWD] [,CNV] [,DIR][,FKS][,FWD] [,GEN][,KEQ][,KEY][,KGE][,LOC] [,LRD][,MVE][,NSP][,NUP][,SEQ] [,SKP][,SYN][,UPD])| ACB=*address*| AREA=*address*| AREALEN=*abs expression*| ARG=*address*| ECB=*address*| FDBK=*abs expression*| KEYLEN=*abs expression*| MSGAREA=*address*| MSGLEN=*abs expression*| NXTRPL=*address*| RBA=*abs expression*| RECLEN=*abs expression*| RPLLEN=*abs expression*| TRANSID=*abs expression*}

The subparameters of the TESTCB macro can be expressed as absolute numeric expressions, as character strings, as codes, as expressions that generate valid relocatable A-type address constants, in register notation, as S-type address constants, and as indirect S-type address constants. "Subparameters with GENCB, MODCB, SHOWCB, and TESTCB" on page 5, further defines these operand expressions.

label
specifies 1 to 8 characters that provide a symbolic address for the TESTCB macro.

RPL=*address*
specifies the address of the request parameter list whose information you want to test. You may omit it only if you are testing the length of a request parameter list (RPLLEN=*number*). (All request parameter lists have the same length.)

ERET=*address*
specifies the address of a routine to which VSAM gives control if an error occurs and VSAM is unable to test for the specified condition. For example, testing AVSPAC in an access method control block for an unopened data set would fail. VSAM indicates in register 15 whether it could do the test and, if not, indicates in register 0 the reason it could not. (The reasons are discussed under "Control block manipulation macro return and reason codes" on page 118.) A failure trying to execute TESTCB indicates a basic logical problem in the processing program, so the error routine would probably issue an abend. If it lets the program continue, it must branch to the continuation point itself, and not return to VSAM.

AIXFLAG=AIXPKP
specifies that prime-key pointers are used rather than RBAs.

AIXPC=*abs expression*
specifies the pointer count.

FTNCD=*abs expression*
specifies whether the upgrade set is correct or may have been modified by a request. These codes are described under "Component codes (RPLCMPON)" on page 122.

IO=COMPLETE
specifies that a test is made to determine whether an asynchronous request has been completed. (When you issue a CHECK macro, you suspend processing until a request has been completed.)

OPTCD=([,ADR][,ARD][,ASY][,BWD] [,CNV]
[,DIR][,FKS][,FWD][,GEN][,KEQ]
[,KEY][,KGE][,LOC][,LRD][,MVE]
[,NSP][,NUP][,SEQ]
[,SKP][,SYN][,UPD]
specifies that a test is to be made to determine what subparameter or combination of subparameters is being used for the request. See Table 3 on page 72 for a description of these subparameters.

Example: TESTCB macro (test a request parameter list)

```
        TESTCB RPL=(3),                                          x
               RECLEN=80

         BE    NOCHNGE

CHANGE   ...              Because record length in the RPL not 80,    x
                          modify length indicator so it is 80.

NOCHNGE  ...              Because record length in the RPL is 80,     x
                          no change required.
```

The TESTCB macro's parameters are:

- RPL specifies that the address of the request parameter list to be tested is contained in register 3.
- RECLEN specifies that the record length indicated in the request parameter list is to be tested to determine whether it is 80.

TESTCB—List form

The format of the list form of TESTCB is:

[*label*]	TESTCB	[{ACB|EXLST|RPL}=*address*] [,ERET=*address*] *keyword*={*address*|*name*|*abs expression*|*option*},... ,MF={L|(L,*address*[,*label*])} [,OBJECT={DATA|INDEX}]

If the execute form of TESTCB is used and EXLST is used as a keyword to be processed, the block must be identified by ACB=.

TESTCB—Execute form

The format of the execute form of TESTCB is:

[*label*]	TESTCB	[{ACB|EXLST|RPL}=*address*] [,ERET=*address*] *keyword*={*address*|*name*|*abs expression*|*option*},... ,MF=(E,*address*) [,OBJECT={DATA|INDEX}]

Rule: If the execute form of TESTCB is used and EXLST is used as a keyword to be processed, the block must be identified by ACB=.

TESTCB—Generate form

The format of the generate form of TESTCB is:

[*label*]	TESTCB	[{ACB|EXLST|RPL}=*address*] [,ERET=*address*] *keyword*={*address*|*name*|*abs expression*|*option*},... ,MF=(G,*address*[,*label*]) [,OBJECT={DATA|INDEX}]

VERIFY—Synchronize end of data

Use the VERIFY macro to synchronize end-of-data.

VERIFY is not supported for z/OS UNIX files and returns an error if specified for these files.

The format of the VERIFY macro is:

[*label*]	VERIFY	RPL=*address* [,ACTION=REFRESH]

label
specifies 1 to 8 characters that provide a symbolic address for the VERIFY macro.

RPL=*address*
specifies the address of the request parameter list defining this VERIFY request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

The following parameter and subparameter are required for VERIFY:

In the RPL, OPTCD=(CNV,...) must be specified.

ACTION=REFRESH
specifies the VSAM control blocks that are to be updated from the catalog after an attempt is made to verify the high-used RBA. For a data set that has been extended, VERIFY with ACTION=REFRESH causes an update to the control block structure, reflecting the new extents.

If you do not specify ACTION=REFRESH for an extended data set, you must close the data set and reopen it to obtain new extent information before you can verify it.

Any attempt to issue the VERIFY macro against a linear data set (LDS) results in a logical error (return code 253 in the feedback field of the RPL).

RLS does not support VERIFY because RLS maintains the end of data set information in the control blocks.

After verifying a data set, positioning must be established with a POINT macro for sequential processing or with a GET macro with RPL OPTCD=DIR.

WRTBFR—Write buffer

If you are using local or global shared resources, you can use the WRTBFR macro to write a buffer.

The format of the WRTBFR macro is:

[*label*]	WRTBFR	RPL=*address* ,TYPE={ALL|CHK|DRBA|DS|LRU(*percent*)|TRN}

label
specifies 1 to 8 characters that provide a symbolic address for the WRTBFR macro.

RPL=*address*
specifies the address of the request parameter list that defines the WRTBFR request. An RPL need not be built especially for the WRTBFR. WRTBFR may use an inactive RPL that defines other requests (GET, PUT, and so forth) for a data set using the resource pool. The following RPL parameters have meaning for WRTBFR:

ACB=*address*
ARG=*address*
For TYPE=DRBA, the address of a 4-byte field that contains the RBA to be located and written. For compressed data sets, the RBA of another record or the address of the next record in a buffer cannot be determined using the length of the current record or the length of the record provided to VSAM.

For extended addressing, the address of an 8-byte field that contains the RBA to be located and written.

ECB=*address*

OPTCD={ASY|SYN}
WRTBFR can be issued synchronously (SYN) or asynchronously (ASY). A CHECK or ENDREQ must be issued to synchronize an asynchronous WRTBFR request.

TRANSID=*abs expression*
specifies a number from 0 to 31.

All other RPL parameters are ignored. RPLs are assumed not to be chained.

If the ACB to which the RPL is related has MACRF=GSR, the program issuing WRTBFR must be in supervisor state with protection key 0 to 7.

TYPE={ALL|CHK|DRBA|DS|LRU(*percent*)|TRN}
specifies which buffers are to be written.

Rule: Before using WRTBFR TYPE=CHK|DRBA|TRN, be sure to release all buffers. VSAM defers processing until all buffers are released. For details about releasing buffers, see *z/OS DFSMS Using Data Sets*.

ALL
specifies that all modified unwritten index and data buffers in each buffer pool in the resource pool are to be written. All buffers with physical errors from WRTBFR are invalidated. Closing all the data sets that use a resource pool causes the same buffers to be written.

CHK
is the same as TRN (below), but, if any error occurs in writing buffers, transaction IDs continue to be associated with the buffers. If there are no errors, transaction IDs are no longer be associated with the buffers. WRTBFR TYPE=CHK can be used by a checkpoint routine to record checkpoint information and leave buffers for which an error occurred as they were for continued processing.

DRBA
specifies that one of the data set's data buffers is to be written. The buffer to be written is identified with the RBA pointed to by the RPL ARG address.

DS
specifies that, for the data set defined by the ACB to which the WRTBFR's RPL is related, all modified unwritten index and data buffers are to be written and all buffers (including the Hiperspace buffers) are to be marked empty, that is, invalidated. **Therefore, WRTBFR TYPE=DS should be issued only after all VSAM requests for the data set have been quiesced. Otherwise, the results might be unpredictable.**

LRU(*percent*)
specifies that some of the modified buffers in each buffer pool in the resource pool are written. The percent is the percentage of buffers in each pool that are examined for possible writing. The least recently used buffers are examined. (If percent is coded in register notation, only registers 1 and 13 may not be used.)

TYPE=LRU is used for writing some modified buffers, without respect to a particular data set or transaction ID, to ensure that buffers are available for GET requests (without having to wait for buffers to be written).

TRN
specifies that all buffers in a buffer pool that are modified by requests with the transaction ID that is specified in the WRTBFR's RPL are to be written. Transaction IDs are no longer associated with these buffers if WRTBFR completes successfully, or if a physical error occurs. Otherwise, the transaction buffers are still associated with these buffers.

Chapter 3. VSAM macro return and reason codes

This chapter describes the return codes and reason codes that are generated by the VSAM macros that are used to open and close data sets, manage VSAM control blocks, and issue record management requests.

VSAM sets the return codes in register 15. (For information on register usage conventions, see "Rules for register usage" on page xxi.) These return codes are paired with reason codes that are set in the access method control block (ACB) and the request parameter list (RPL). Reason codes that are set in the ACB indicate open or close errors. Reason codes that are set in the RPL indicate record management errors.

This manual lists return codes and reason codes as decimal and hexadecimal values. The decimal value is shown first, followed by the hexadecimal value in parentheses. Format descriptions and examples of each macro are shown in Chapter 2, "VSAM macro descriptions and examples," on page 5. Some VSAM reason codes, which are used for diagnostic purposes, are shown in *z/OS DFSMSdfp Diagnosis*.

OPEN return and reason codes

When your program receives control after issuing an OPEN macro, the return code in register 15 indicates if all the data sets were opened successfully (see Table 7 on page 107).

Table 7. Return codes in register 15 after OPEN

Return Code	Meaning
0(X'0')	All data sets were opened successfully.
4(X'4')	All data sets were opened successfully, but one or more attention messages were issued (reason codes in the ACBERFLG field of the ACB less than X'76'). Non-VSAM OPENs do not issue attention messages, so a return code 4 does not occur for non-VSAM data sets. Only VSAM OPENs issue attention messages resulting in a return code of 4.
8(X'8')	At least one data set (VSAM or non-VSAM) was not opened successfully; the access method control block was restored to the contents it had before the OPEN was issued; or, if the data set was already open, the access method control block remains open and usable and is not changed.
12(X'C')	A non-VSAM data set was not opened successfully when a non-VSAM and a VSAM data set were being opened at the same time. The non-VSAM data control block was not restored to the contents it had before the OPEN was issued (and the data set cannot be opened without restoring the control block).
16(X'10')	One or more of the access method control blocks (ACBs) specified the RLS option but the system has not been set up for RLS (the SMSVSAM server address space is not available). For other DCBs and ACBs any condition described by other return codes is possible.

If register 15 contains a nonzero return code, use the SHOWCB macro to display the corresponding reason code. The SHOWCB macro displays the error field in each access method control block specified by the OPEN macro. (See "SHOWCB—Display fields of an access method control block" on page 84.)

Table 8 on page 108 lists the reason codes that may appear in this error field.

Table 8. OPEN reason codes in the ACBERFLG field of the ACB

Reason Code	Meaning
0(X'0')	One of the following conditions exists: • VSAM is processing the access method control block for some other request. • The access method control block address is invalid.
72(X'48')	One of the following errors occurred (a warning): • A non-RLS or non-DFSMStvs OPEN for input was successful against a sphere that was already in a lost locks or retained locks state. • A non-RLS or non-DFSMStvs OPEN for output was successful against a sphere that was already in a lost locks or retained locks state because a NONRLSUPDATE was in effect.
76(X'4C')	The interrupt recognition flag (IRF) was detected for a data set opened for input processing. This indicates that DELETE processing was interrupted. The structure of the data set is unpredictable; the access method services DIAGNOSE command can be used to check the data set for structural errors. For a description of the DIAGNOSE command, see *z/OS DFSMS Access Method Services Commands*.
82(X'52')	With OPEN for output processing of a base data set, an associated upgrade AIX is also being opened, but the OPEN requested an unexpected LSR pool via the ACB SHRPOOL value. See message IEC161I 042 for more details. The open was successful.
88(X'58')	A previous extend error has occurred during EOV processing of the data set. For MACRF=RLS, reset processing of "delete vol" has received an error.
92(X'5C')	Inconsistent use of CBUF processing. Sharing options differ between index and data components.
96(X'60')	An unusable data set was opened for input.
100(X'64')	An OPEN found an empty alternate index that is part of an upgrade set.
101(X'65')	For MACRF=RLS, the sphere that was opened is in lost locks state. The open was successful.
102(X'66')	For MACRF=RLS, the sphere is in a non-RLS update permitted state. The open was successful.
103(X'67')	For RLS, the sphere that was opened is in both a lost locks state and non-RLS update permitted state. The open is successful. For DFSMStvs, the open succeeded, but one of the following conditions was detected: • DFSMStvs is quiescing due to an I/O error on one of the system logs (the undo log or the shunt log). • The forward recovery log is quiescing due to an I/O error. Processing continues without the forward recovery log. • A failure occurred during an attempt to write a tie-up record to the forward recovery log. Processing continues without the forward recovery log. • The log of logs is quiescing due to an I/O error. Processing continues without the log of logs. • A failure occurred during an attempt to write a tie-up record to the log of logs. Processing continues without the log of logs.
104(X'68')	The time stamp of the volume where the data set is stored does not match the system time stamp in the data set's catalog record. This indicates extent information in the catalog record might not agree with the extents indicated in the volume's VTOC.
108(X'6C')	The time stamps of a data component and an index component do not match. This indicates that either the data or the index has been updated separately from the other.

Table 8. OPEN reason codes in the ACBERFLG field of the ACB (continued)

Reason Code	Meaning
110(X'6E')	JRNAD exit was not specified on the first ACB opened for the data set. Processing continues without journaling.
116(X'74')	The data set was not properly closed. The data set high-used RBA has not been verified. Records might be missing or duplicated. A previous VSAM program might have abnormally terminated. You should verify that all of the expected records are in the data set. If you ignore the message and try to process the data set, the results are unpredictable. The catalog will be updated when the data set has been successfully opened for output and then successfully closed. You can determine if this error occurred on opening an empty data set by using the SHOWCB macro instruction. The SHOWCB macro instruction is described in *z/OS DFSMS Macro Instructions for Data Sets*. For additional information on recovery processing, see *z/OS DFSMS Using Data Sets*.
118(X'76')	The data set was not properly closed. The data set high-used RBA has been successfully verified. Records may be missing or duplicated. A previous VSAM program may have abnormally ended. You should verify that all of the expected records are in the data set. The catalog will be updated when the data set has been successfully opened for output and then successfully closed. For additional information on recovery processing, see *z/OS DFSMS Using Data Sets*.
128(X'80')	DD statement for this access method control block is missing or invalid.
130(X'82')	Open connect is not allowed at this time.
131(X'83')	An error was detected by VSAM for a media manager CONNECT.
132(X'84')	One of the following errors occurred: • Not enough storage was available for work areas. • The required volume could not be mounted. • A system logic error occurred while VSAM was accessing the job file control block (JFCB). • The format-1 DSCB or the catalog cluster record is invalid. • The user-supplied catalog name does not match the name on the entry. • The user is not authorized to open the catalog as a catalog. • For DFSMStvs: – Unable to connect to the forward recovery log – Unable to write tie-up record to the forward recovery log – Data set cannot be opened because it needs to be forward recovered. – DFSMStvs processing is not available. – For a data set that was previously accessed for Permit Non-RLS Update (PNRLU) processing, an error occurred attempting to write the PNRLU record to the undo log
133(X'85')	Delete Volume processing for RESET(MACRF=RST) failed during open. The DDNAME needs to be freed and re-allocated to the data set.
134(X'86')	Invalid UCB address for UCB address conversion.

Table 8. OPEN reason codes in the ACBERFLG field of the ACB (continued)

Reason Code	Meaning
136(X'88')	Not enough virtual storage space is available in your program's address space for work areas, control blocks, or buffers.
138(X'8A')	A 24-bit UCB address is required for Volume Mount but a 31-bit UCB address was passed.
140(X'8C')	The catalog indicates this data set has an invalid physical record size.
144(X'90')	Uncorrectable I/O error occurred while VSAM reading or writing catalog record.
145(X'91')	An uncorrectable error occurred in the VSAM volume data set (VVDS).
148(X'94')	No record for the data set to be opened was found in the available catalogs, or an unidentified error occurred while VSAM was searching the catalog. For the catalog return code, see system message IDC3009I. For a description of this message, see *z/OS MVS System Messages, Vol 6 (GOS-IEA)*. For z/OS UNIX files, the requested file does not exist.
152(X'98')	Authorization checking failed for one of the following reasons: • The password specified in the access method control block for a specified level of access does not match the password in the catalog for that level of access. • The job is not authorized for the KEYLABEL. • RACF denied access. For the catalog return code, see system message IDC3009I in job output. For a description of this message, see *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

Table 8. OPEN reason codes in the ACBERFLG field of the ACB (continued)

Reason Code	Meaning
160(X'A0')	The operands specified in the ACB or GENCB macro are inconsistent either with each other or with the information in the catalog record. One of these conditions has been detected: • For option ACBRST – Path processing – LSR or GSR • For option ACBICI – LSR or GSR – Key-sequenced data set – Path processing – Sequence set with data – Replicated index – Block size not equal to CI size – Extended format data set • For option ACBUBF – LSR or GSR – ACBCNV not specified – ACBKEY specified – ACBADR specified • For option ACBSDS – LSR or GSR – Path processing – Upgrade processing • For option ACBCBIC – LSR or GSR – ACBICI not specified • For option RLS, an invalid option has been specified. See the message for further information. • For miscellaneous options – Buffer space specified but the amount is too small to process the data set – Volume not mounted – Trying to open an empty data set for input • For a z/OS UNIX file, an invalid option or operand has been specified – ACBCNV or ACBKEY – ACBSKP – ACBICI – LSR, GSR, or RLS – ACBSTRNO > 1.

Table 8. OPEN reason codes in the ACBERFLG field of the ACB (continued)

Reason Code	Meaning
161(X'A1')	The data set attributes are not compatible with the data set accessed on a volume with the read-only attribute. One of the following conditions exist: • Full read/write integrity was specified with DISP=SHR. • Full read/write integrity was specified with SHAREOPTIONS(1 3). • The CONTROLINTERVALSIZE will cause some CIs to cross a track boundary.
162(X'A2')	The multi-volume data set cannot include both read-write and read-only volumes.
163(X'A3')	The encrypted data set has an encryption key which is archived and only decryption is allowed.
164(X'A4')	An uncorrectable I/O error occurred while VSAM was reading the volume label.
165(X'A5')	VSAM does not support encrypted non-extended format data sets
166(X'A6')	VSAM does not support GRS buffering for encrypted data sets
167(X'A7')	For MACRF=RLS, open or close processing received an abend while processing the request.
168(X'A8')	The data set was not available for the type of processing that you specified. Or, an attempt was made to open a reusable data set with the reset option while another user had the data set open. The data set might have the INHIBIT attribute specified. The data set cannot be opened for CBUF processing because it was already opened for non-CBUF processing. Or, the data set has conflicting CBUF attributes for the data and index components of the ACB. For MACRF=RLS, an attempt was made to access a data set with NSR/LSR/GSR and the data set is currently accessed by RLS or DFSMStvs, or vice versa. Or, an attempt was made to access the data set with NSR/LSR/GSR and the data set is in lost or retained locks state. For a z/OS UNIX file, the file has a file type that is not supported (for example, directories are not supported).
169(X'A9')	KEYLABEL has wrong encryption type.
170(X'AA')	For RLS, an ACB specified a SUBSYSNM name, which is already registered to a previous server instance.
171(X'AB')	For MACRF=RLS, required CF cache is unavailable from this system.
172(X'AC')	For RLS, CF Cache structure failed.
173(X'AD')	For RLS, required CF cache structure is in a quiescing or quiesced state.
174(X'AE')	One of the following errors occurred: • For MACRF=RLS, when DFSMStvs is not active on the system, SUBSYSNM was not specified in the ACB and an attempt was made to open a data set for output to a recoverable sphere. (In that situation, if DFSMStvs is active on the system, the data set would be opened for DFSMStvs without this error.) • The LOG parameter was changed to LOG(NONE) while recovery was pending against the data set.
175(X'AF')	For RLS, locks have been lost. This is an attempt by a new sharing SUBSYSNM to access a data set for which not all recovery has completed. The open is not successful.
176(X'B0')	Data Set is encrypted but user did not specify encryption was allowed

Table 8. OPEN reason codes in the ACBERFLG field of the ACB (continued)

Reason Code	Meaning
177(X'B1')	For RLS or DFSMStvs, the open is rejected because the sphere is marked VSAM quiesced.
178(X'B2')	For MACRF=RLS, the open is rejected. The sphere is VSAM-quiescing and this is an attempt by a new application.
179(X'B3')	For RLS, the open is rejected. The sphere is VSAM-quiescing in preparation for a data set copy, or a data set copy is in progress.
180(X'B4')	A catalog specified in JCL either does not exist or is not open, and no record for the data set to be opened was found in any other catalog.
181(X'B5')	For MACRF=RLS, the DISP value specified is not consistent with the DISP value specified by another application that has opened this data set for RLS access. Either this application is requesting DISP=SHR while another application holds DISP=OLD or vice-versa.
182(X'B6')	For MACRF=RLS, the SMSVSAM server is not available.
183(X'B7')	For RLS open, invalid backup while open (BWO) flags in the catalog.
184(X'B8')	An uncorrectable I/O error occurred while VSAM was completing an I/O request.
186(X'BA')	An error was returned from ICSF.
187(X'BB')	Encryption key has changed since the data set was loaded.
188(X'BC')	The data set that is indicated by the access method control block is not of the type that can be specified by an access method control block. Or the access method control block (ACB) has already been opened or closed.
189(X'BD')	The Exit List (EXLST) is invalid because the length is incorrect.
190(X'BE')	An invalid hi-allocated RBA was found in the catalog entry for this data set. The catalog entry is bad and will have to be restored.
192(X'C0')	An unusable data set was opened for output.
193(X'C1')	The interrupt recognition flag (IRF) was detected for a data set opened for output processing. This indicates that DELETE processing was interrupted. The structure of the data set is unpredictable. The access method services DIAGNOSE command may be used to check it for structural errors. For a description of the DIAGNOSE command, see *z/OS DFSMS Access Method Services Commands*.
194(X'C2')	An open of the data component of a compressed format key-sequenced data set is not allowed. For MACRF=RLS, an attempt was made to open an alternate index cluster or an individual component of a KSDS data set. KSDS components cannot be opened for RLS processing.
195(X'C3')	For MACRF=RLS, the SMS Storage Class does not specify a coupling facility CACHESET name.
196(X'C4')	Access to data was requested via an empty path. For RLS: • Access to data was requested through an empty path. • Attempt to access a VSAM data set for RLS processing via an Alternate Index which is not part of the Upgrade Set.
197(X'C5')	Catalog indicated RLS recovery required but user's ACB did not specify recovery processing.
198(X'C6')	For RLS, an open is rejected because a volume quiesce is in progress or a required volume is marked as "quiesced".

Table 8. OPEN reason codes in the ACBERFLG field of the ACB (continued)

Reason Code	Meaning
200(X'C8')	The format-4 DSCB indicates that the volume is unusable.
201(X'C9')	For RLS, the sphere is not currently assigned to a CF cache and there are no CF caches available from this system which could be assigned to the sphere.
202(X'CA')	For RLS, SUBSYSNM violation. The SUBSYSNM name specified is different from the subsystem name registered for this address space.
203(X'CB')	For RLS, JRNAD Exit requested for ACB being opened for RLS processing.
204(X'CC')	The ACB MACRF specification is GSR and caller is not operating in protect key 0 to 7. Or, ACB MACRF specification is CBIC (Control Blocks in Common) and caller is not operating in supervisor state with protect key 0 to 7.
205(X'CD')	The ACBCATX option or VSAM volume data set open was specified and the calling program was not authorized.
206(X'CE')	For MACRF=RLS, the LOG parameter that is associated with the base cluster is undefined.
207(X'CF')	RLS SUBSYSNM name contains invalid characters.
208(X'D0')	System logic error.
209(X'D1')	RLS or DFSMStvs open internal logic error detected.
210(X'D2')	RLS or DFSMStvs open requested for non-SMS-managed data set.
211(X'D3')	A previous MSGIGW405I has been issued. All opens issued for the sphere on the system must be closed and the sphere examined for possible corruption. The sphere may then be opened successfully.
212(X'D4')	The ACB MACRF specification is GSR or LSR and the data set requires load mode processing.
214(X'D6')	For DFSMStvs, the maximum logical record length for the data set is larger than the length that DFSMStvs supports for logging.
216(X'D8')	The ACB MACRF specification is GSR or LSR and the key length of the data set exceeds the maximum key length specified in BLDVRP.
220(X'DC')	The ACB MACRF specification is GSR or LSR and the data set's control interval size exceeds the size of the largest buffer specified in BLDVRP.
224(X'E0')	Improved control interval processing is specified and the data set requires load mode processing.
228(X'E4')	The ACB MACRF specification is GSR or LSR and the VSAM shared resource table (VSRT) does not exist (no buffer pool is available).
229(X'E5')	OPEN failed because a BLDVRP or DLVRP is already in progress. A retry of the OPEN is suggested.
230(X'E6')	OPEN failed because the maximum number of alternate indexes (255) has been exceeded.
231(X'E7')	OPEN failed because the maximum number of VSAM control blocks has been exceeded.
232(X'E8')	Reset was specified for a non-reusable data set and the data set is not empty.
236(X'EC')	System logic error.
240(X'F0')	Format-4 DSCB and volume timestamp verification failed during volume mount processing for output processing.
244(X'F4')	The volume containing the catalog recovery area was neither mounted nor verified for output processing.

Table 8. OPEN reason codes in the ACBERFLG field of the ACB (continued)

Reason Code	Meaning
245(X'F5')	An attempt was made to open a compressed format data set without sufficient hardware, ESCON channels and concurrent copy capable control units, or a compressed format device was required.
246(X'F6')	The compression management services open or close function failed.
247(X'F7')	An error occurred while retrieving the dictionary token from the extended format cell.
250(X'FA')	DSAB match not found.

VSAM also writes a message to the operator console and the programmer's listing further explaining the error. For a listing of VSAM messages, see *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

CLOSE return and reason codes

When your program receives control after it has issued a CLOSE macro, a return code in register 15 indicates whether all VSAM data sets were closed successfully (see Table 9 on page 115).

Table 9. Return Codes in Register 15 After CLOSE

Return Code	Meaning
0(X'0')	All data sets were closed successfully.
4(X'4')	At least one data set (VSAM or non-VSAM) was not closed successfully.

If register 15 contains 4, use SHOWCB to display the ERROR field in each access method control block to determine if a VSAM data set was not closed successfully and the reason it was not. See "SHOWCB—Display fields of an access method control block" on page 84. Table 10 on page 115 gives the reason codes the ERROR field may contain following close processing.

Table 10. CLOSE Reason Codes in the ACBERFLG Field of the ACB

Reason Code	Meaning
0(X'0')	No error (set when register 15 contains 0).
4(X'4')	The data set indicated by the access method control block is already closed.
129(X'81')	CLOSE TYPE=T was issued for a VSAM data set that is not open for VSAM processing.
132(X'84')	An uncorrectable I/O error occurred while VSAM was reading the job file control block (JFCB).
136(X'88')	Not enough virtual storage was available in your program's address space for a work area for close processing.
144(X'90')	An uncorrectable I/O error occurred while VSAM was reading or writing a catalog record.
145(X'91')	An uncorrectable error occurred in the VSAM volume data set (VVDS).
148(X'94')	An unidentified error occurred while VSAM was searching the catalog. For a z/OS UNIX file, an unidentified error occurred.
167(X'A7')	For RLS, abend occurred during open or close processing.
170(X'AA')	For RLS, the required CF Cache is unavailable from this system.
171(X'AB')	The close was successful, but the data set is in use by a unit of recovery that is still inflight.

Table 10. CLOSE Reason Codes in the ACBERFLG Field of the ACB (continued)

Reason Code	Meaning
172(X'AC')	Close was successful, but DFSMStvs was unable to write a close record to the log of logs, the forward recovery log, or both.
184(X'B8')	An uncorrectable I/O error occurred while VSAM was completing outstanding I/O requests. For a z/OS UNIX file, an error occurred while flushing output data or when disconnecting from the file.
185(X'B9')	LSR/GSR - Error in WRTBFR: I/O for data set not quiesced before WRTBFR TYPE=DS during close processing.
188(X'BC')	The data set indicated by the ACB is not the type that may be specified by an ACB. For RLS, an invalid ACB address is specified for close processing.
236(X'EC')	System logic error because the function no longer is supported.
246(X'F6')	A call to compression management services (CMS) failed.

In addition to these reason codes, VSAM writes a message to the operator's console and the programmer's listing further explaining the error. For a description of messages, see *z/OS MVS System Messages, Vol 6 (GOS-IEA)*.

OPEN/CLOSE message area for multiple reason or attention messages

This section does not apply to RLS processing. The MAREA and MLEN parameters are ignored by RLS processing.

During the execution of a non-RLS open or close, more than one error condition may be detected. However, the ACB error flag field can accommodate only one attention or error condition. To receive multiple error or attention conditions, specify an optional message area. VSAM uses this optional message area to accumulate error messages from an open or close.

The system can supply multiple messages if you specify nonzero values in the MAREA and MLEN parameters of the ACB. If MAREA or MLEN is either not specified or zero, no error or attention information is stored in the message area. The ACB error flag field is the only indication of error or attention conditions. If MAREA and MLEN are specified and the message area is too small to accommodate all messages, the last incoming messages are dropped. However, you will receive an indication of the number of attention conditions and messages that occurred.

The message area provided by VSAM is divided into two parts:

- The message area header
- The message list

Message area header

The message area header contains statistical, pointer, and general information. Its contents are unrelated to the individual messages. Figure 3 on page 117 shows the format of the message area header.

Byte 0
Flag Byte

bit 0=1
Full message area header has been stored.

bit 0=0
Only flag byte of message area header has been stored. (Implies that no messages have been stored.)

bits 1-7
Reserved (set to binary zeros).

Bytes 1-2
Length of message area header (includes flag byte and length byte).

Byte 3
Request type code:

X'01'
OPEN

X'02'
CLOSE without TYPE=T

X'03'
CLOSE TYPE=T

Bytes 4-11
ddname used for ACB.

Bytes 12-13
Total number of messages (error or attention conditions) issued by open or close processing.

Bytes 14-15
Number of messages stored by open or close processing in message area.

Bytes 16-19
Address of message list of first message in message area.

Figure 3. Format of the Message Area Header

The function of the ACB error flag field remains unchanged whether this optional message area is specified. At the end of an open or close, this field contains either X'00' (indicating no error or attention condition occurred) or a nonzero code. The ACB error flag byte contains the nonzero open or close reason code corresponding to the error or attention condition that occurred with the highest severity.

Message area header information is stored only when an attention or error condition is detected. (That is, when the ACB error flag field is set to a nonzero value.) The header information consists of the flag byte only if the message area length (MLEN) is not large enough to accommodate the full message area header. In this case, bit 0 of the flag byte is zero.

Before accessing the message header information (bytes 1 through 19), test byte 0 to see if more information is stored. If MLEN=0, no header information is stored, not even the flag byte. If the full message area header is stored, bytes 1 and 2 contain its actual length. Your program should be sensitive to this length when interrogating the message area header.

Message list

The message list contains individual messages that correspond to detected attention or error conditions. Bytes 16 through 19 of the message area header contain the location of the message list within the message area. If the message area header is not stored completely, (bit 0 of byte 0 is 0), the location of the message list is not provided.

In the message list, individual messages are stored as a contiguous string of variable-length records. Bytes 14 and 15 of the message area header contain the number of messages stored. Check for a nonzero stored message count before investigating the message list. However, messages may not be stored even if the ACB error flag byte contains a nonzero value and the message area header bit 0 of byte 0 is 1. For

example, no messages will be stored if MLEN is not large enough to allow at least one message to be stored.

The following shows the format of the individual messages.

Bytes 0-1
Length of message (including these 2 bytes).

Byte 2
ACB error flag code corresponding to the error or attention condition represented by this message.

Byte 3
Function type code:

Specifies which dsname, if any, is stored in bytes 4 through 47 of the message:

X'00'
No dsname stored. Bytes 4-47 of the message contain binary zeros. The error attention condition is not clearly related to a component, or VSAM was unable to identify or obtain the cluster name of the component in error. This code is used only if the ddname of the ACB does not identify a valid DD statement, or VSAM was unable to obtain the dsname contained in the DD statement.

X'01'
dsname contained in DD statement is stored. The error or attention condition is not clearly related to a component, or VSAM was unable to identify or obtain the cluster name of the component in error.

X'02'
dsname (cluster name) of base cluster stored. Error occurred during an open or close for base cluster.

X'03'
dsname (cluster name) of alternate index component stored. Error occurred during open or close processing for alternate index component.

X'04'
dsname (cluster name) of member of upgrade set stored. Error occurred during open or close for this member of the upgrade set.

Bytes 4-47
Binary zeros (function type code=X'00') or a dsname as described by byte 3.

Bytes 0 and 1 of each message specify its actual length. Because messages vary in length, you need to know the actual length of each message to do your processing.

Byte 2 of the message contains the ACB error flag code; it does not indicate a dsname has been stored. Depending on the condition that raised the ACB error flag code, either no dsname or different types of dsnames (DD, base cluster, alternate index, or upgrade set member) may be stored. (The same condition may be detected both when opening the base cluster and when opening a member of the upgrade set. For example, an I/O error may occur when trying to obtain the dsname for the component in error.)

Bytes 4 through 47 of the message can contain a dsname, but not specify its type.

Only byte 3 of the message specifies if a dsname was stored and, if so, its type.

Control block manipulation macro return and reason codes

The GENCB, MODCB, SHOWCB, and TESTCB macros can be executed (unlike the ACB, EXLST, and RPL macros). They cause control to be given to VSAM to perform the indicated task. VSAM indicates if the task was completed by a return code in register 15 (see Table 11 on page 118).

Table 11. Return Codes in Register 15 After Control Block Manipulation Macros

Return Code	Meaning
0(X'0')	Task completed.

Table 11. Return Codes in Register 15 After Control Block Manipulation Macros (continued)

Return Code	Meaning
4(X'4')	Task not completed.
8(X'8')	An attempt was made to use the execute form of a macro to modify a keyword that is not in the parameter list. (See "Use of list, execute, and generate forms of VSAM macros" on page 6.)

You can cause an error if you specify the operands incorrectly.

When register 15 contains 4, register 0 contains a reason code indicating why VSAM could not perform the task. If you construct the parameter list, register 0 can contain reason codes 1, 2, 3, 10, 14, 20, and 21.

Table 12 on page 119 describes each reason code returned in register 0.

Table 12. GENCB, MODCB, SHOWCB, and TESTCB Reason Codes Returned in Register 0

Reason Code	Applicable Macros"1" on page 120	Reason VSAM Could Not Perform the Task
1(X'1')	G,M,S,T	The request type (generate, modify, show, or test) is invalid.
2(X'2')	G,M,S,T	The block type (access method control block, exit list, or request parameter list) is invalid.
3(X'3')	G,M,S,T	One of the keyword codes in the parameter list is invalid.
4(X'4')	M,S,T	The block at the address indicated is not of the type you indicated (access method control block, exit list, or request parameter list).
5(X'5')	S,T	Access method control block fields were to be shown or tested, but the data set is not open or it is not a VSAM data set.
6(X'6')	S,T	Access method control block information about an index was to be shown or tested, but no index was opened with the data set.
7(X'7')	M,S	An exit list was to be modified, but the list was not large enough to contain the new entry. Or, an exit was to be modified or shown but the specified exit wasn't in the exit list. (With TESTCB, if the specified exit address is not present, you get an unequal condition when you test for it.)
8(X'8')	G	There is not enough virtual storage in your program's address space to generate the access method control blocks, exit lists, or request parameter lists and no work area outside your address space was specified.
9(X'9')	G,S	The work area specified was too small for generation or display of the indicated control block or fields.
10(X'A')	G,M	With GENCB, exit list control block type was specified and you specified an exit without giving an address. With MODCB, exit list control block type was specified and you specified an exit without giving an address. In this case, either active or inactive must be specified, but load cannot be specified.
11(X'B')	M	Either (1) a request parameter list was to be modified, but the request parameter list defines an asynchronous request that is active (that is, no CHECK or ENDREQ has been issued on the request) and thus cannot be modified; or (2) MODCB is already issued for the control block, but has not yet completed.

Table 12. GENCB, MODCB, SHOWCB, and TESTCB Reason Codes Returned in Register 0 (continued)

Reason Code	Applicable Macros[“1” on page 120]	Reason VSAM Could Not Perform the Task
12(X'C')	M	An access method control block was to be modified, but the data set identified by the access method control block is open and cannot be modified.
13(X'D')	M	An exit list was to be modified, and you attempted to activate an exit without providing a new exit address. Because the indicated exit list does not contain an address for that exit, your request cannot be honored.
14(X'E')	G,M,T	One of the option codes (for MACRF, ATRB, or OPTCD) has an invalid combination of option codes specified (for example, OPTCD=(ADR,SKP)).
15(X'F')	G,S	The work area specified did not begin on a fullword boundary.
16(X'10')	G,M,S,T	A VTAM® keyword or subparameter was specified but the AM=VTAM parameter was not specified. AM=VTAM must be specified to process a VTAM version of the control block.
19(X'13')	M,S,T	A keyword was specified that refers to a field beyond the length of the control block located at the indicated address. (For example, a VTAM keyword is specified, but the control block it points to is a shorter, non-VTAM block.)
20(X'14')	S	Keywords were specified which apply only if MACRF includes LSR or GSR.
21(X'15')	S,T	The block to be displayed or tested does not exist because the data set is a dummy data set.
22(X'16')	S	AM=VTAM was specified and the RPL FIELDS parameter conflicts with the RPLNIB bit status. Either RPLFIELDS=NIB was specified and the RPLNIB was off, or RPL FIELDS=ARG was specified and the RPLNIB bit was on.
23(X'17')	G	The value specified in the work area length parameter exceeds the 65,535 byte limit.
24(X'18')	S,T	The SMSVSAM server is not available.
25(X'19')	S	LOKEY is not supported for RLS.
26(X'1A')	S,T	This request was issued against an ACB open to a different instance of the SMSVSAM server. The OPEN is no longer valid.
27(X'1A')	G,M,S,T	This request was issued in AR ASC mode, home ASC mode. Or the RLS address space had to be accessed and the request was issued in secondary ASC mode.

Note:

1. G=GENCB, M=MODCB, S=SHOWCB, T=TESTCB

Record management return and reason codes

The following record management macros give return codes and reason codes in the feedback area of the RPL: GET, PUT, POINT, ERASE, VERIFY, CHECK, ENDREQ, GETIX, PUTIX, WRKBFR, SCHBFR, VERIFY, VERIFY REFRESH, and WRTBFR.

The feedback word in the RPL consists of 4 bytes:

Byte
Description

0
Problem determination function (PDF) code. This code is used to locate the point in VSAM record management where a logical error condition is recognized. A description of the returned PDF code is located in the IDARMRCD macro.

1
RPL return code. This code is returned in register 15.

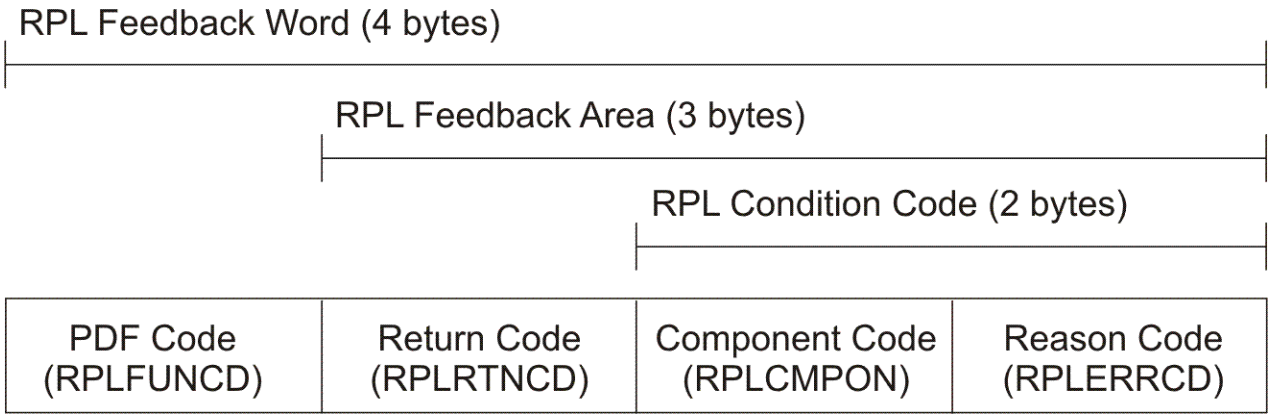
2
Component code. This code specifies the component being processed when the error occurred.

3
Reason code. This code, when paired with the return code in byte 2, specifies the reason for either a successful completion or an error.

Bytes 2 through 4 make up the RPL feedback area. An explanation of the codes that appear in these three bytes follows.

Bytes 3 and 4 make up the RPL condition code. An explanation of this code also follows.

The field name of each byte appears within parentheses in the following figure.



Return codes (RPLRTNCD)

The meaning of the return code depends on whether the processing is asynchronous or synchronous.

Asynchronous request

After you issue an asynchronous request for access to a data set, VSAM sets a return code in register 15 to indicate whether the request was accepted. Table 13 on page 121 describes each return code returned in register 15.

Table 13. Return Code in Register 15 Following an Asynchronous Request

Return Code (RPLRTNCD)	Meaning
0(X'0')	Request was accepted.
4(X'4')	Request was not accepted because the request parameter list indicated by the request (RPL=address) was active for another request.

If the asynchronous request was accepted, issue a CHECK after doing your other processing. This way VSAM can indicate in register 15 whether the request was completed successfully, set a return code in the feedback area, and exit to any appropriate exit routine.

If the request was not accepted, you should either wait until the other request is complete (for example, by issuing a CHECK on the request parameter list) or terminate the other request (using ENDREQ). Then you can reissue the rejected request.

Synchronous request

After a synchronous request, or a CHECK or ENDREQ macro, the return code in register 15 indicates if the request completed successfully. Table 14 on page 122 describes each return code returned in register 15.

Table 14. Return Code in Register 15 Following Synchronous Request

Return Code (RPLRTNCD)	Meaning
0(X'0')	Request completed successfully.
4(X'4')	Request was not accepted because the request parameter list indicated by the request (RPL=address) was active for another request.
8(X'8')	Logical error; specific error is indicated in the RPL feedback area.
12(X'C')	Physical error; specific error is indicated in the RPL feedback area.

Component codes (RPLCMPON)

When a logical or physical error occurs, VSAM uses the RPL component code field to identify the component being processed when the error occurred. VSAM also indicates if the alternate index upgrade set is correct following the request that failed. The component code can be displayed and tested by using the SHOWCB and TESTCB macros. The codes and their meanings are given in Table 15 on page 122.

Table 15. Component Codes Provided in the RPL

Component Code (RPLCMPON)	What Was Being Processed	Upgrade Set Status
0(X'0')	Base cluster	Correct
1(X'1')	Base cluster	May be incorrect
2(X'2')	Alternate index	Correct
3(X'3')	Alternate index	May be incorrect
4(X'4')	Upgrade set	Correct
5(X'5')	Upgrade set	May be incorrect

The component code (byte 3 of the RPL feedback word) and the reason code (byte 4 of the RPL feedback word) make up the two-byte RPL condition code.

Reason codes (RPLERRCD)

The 0, 8, and 12 return codes in register 15 are paired with reason codes in the RPL feedback area.

The reason codes in the RPL feedback area can be examined with the SHOWCB or TESTCB macro. Code your examination routine immediately following the request macro. Logical errors, physical errors, and reaching the end of the data set all cause VSAM to exit to the appropriate exit routine, if one is provided.

Coordinate error checking in your program with your error-analysis exit routines. If they terminate the program, for instance, you do not need to code a check for an error after a request. But, if a routine returns to VSAM to continue processing, you should check register 15 after a request to determine if there was an error. Even when an error is handled by an exit routine, you may want to modify processing because of the error.

Reason code (successful request)

When the request is completed, register 15 indicates the status of the request. A reason code of 0 indicates successful completion. Table 16 on page 123 lists nonzero reason codes and their meanings.

Table 16. Successful Completion Reason Codes in the Feedback Area of the Request Parameter List

Reason Code (RPLERRCD) When Register 15=0(X'0')	Meaning
0(X'0')	Request completed successfully.
4(X'4')	Request completed successfully. For retrieval, VSAM mounted another volume to locate the record. For storage, VSAM allocated additional space or mounted another volume.
8(X'8')	For GET requests, indicates a duplicate alternate key exists (applies only when accessing a data set using an alternate index that allows non-unique keys). For PUT requests, indicates that a duplicate key was created in an alternate index with the non-unique attribute.
9(X'9')	Request completed successfully. For sequential retrieval, when accessing a VSAMDB (KSDS defined with the DATABASE option) with a UNIQUEKEY AIX, a base record (called "document" in VSAMDB) found via a truncated AIX key value has been returned to the caller. **Note:** When an AIX key value is truncated after 251 bytes, other base documents may have AIX key values with the same first 251 characters.
10(X'A')	Request completed successfully. For sequential retrieval, when accessing a VSAMDB (KSDS defined with DATABASE option) with a NONUNIQUEKEY AIX, a base (called "document" in VSAMDB) found via a truncated AIX key value has been returned to the caller. **Note:** (1) As in non-VSAM, the base documents returned from multiple requests against that AIX key value are not sorted by their primary key values. (2) When an AIX key value is truncated after 251 bytes, other base documents may have AIX key values with the same first 251 characters.
12(X'C')	All buffers, except for the buffer just obtained, may have been modified and may need to be written. It is suggested you issue the WRTBFR macro.
16(X'10')	The sequence-set record does not have enough space to allow it to address all the control intervals in the control area that should contain the record. The record was written into a new control area.
20(X'14')	Mass Storage System macros CNVTAD, MNTACQ, and ACQRANGE are no longer supported.
24(X'18')	Buffer found but not modified; no buffer writes performed.
28(X'1C')	Control interval split indicator was detected during an addressed GET NUP request.
32(X'20')	Request deferred for a resource held by the terminated RPL is asynchronous and cannot be restarted. A MRKBFR request is invalid because no candidate buffers can be found. For RLS, there are no locks to retain since no update locks exist for this CICS address space, CICS transaction, or SPHERE.
36(X'24')	Possible data set error condition was detected.
40(X'28')	Possible data set error condition was detected.

Table 16. Successful Completion Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=0(X'0')	Meaning
43(X'2B')	EOV called to retrieve or update the dictionary token in the extended format cell.
44(X'2C')	EOV called to update catalog statistics.
48(X'30')	An error occurred during CA Reclaim. The erase was successful.
52(X'34')	CA reclaim failed due to subtask in key 9. ERASE was successful.
56(X'38')	CA reclaim failed due to no storage. ERASE was successful.
60(X'3C')	CA reclaim failed due to CI #2 not on 2nd level. ERASE was successful.
64(X'40')	CA reclaim failed due to reclaiming high key entry. ERASE was successful.
68(X'44')	For NSR or LSR, not enough buffers were available so VSAM has successfully added additional buffers to complete the request. The request was successful.

Reason code (logical errors)

If a logical error occurs and you have no LERAD routine (or the LERAD exit is inactive), VSAM returns control to your program following the last executed instruction. (See *z/OS DFSMS Using Data Sets* for information on the LERAD routine.)

The return code in register 15 indicates a logical error (8), and the RPL feedback area contains a reason code identifying the error. Register 1 points to the RPL.

Some VSAM reason codes for logical errors, used for diagnosis purposes, are shown in *z/OS DFSMSdfp Diagnosis*.

Table 17 on page 124 lists the feedback area reason codes and their meanings. Some of these reason codes in the figure use the term LUWID in their meaning column. For a CICS application, the LUWID is a CICS transaction identifier. For a batch job, the LUWID is a unique value assigned by RLS to the address space.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
4(X'4')	End of data set found (during sequential or skip sequential retrieval), or the search argument is greater than the high key of the data set. Either no EODAD routine is provided, or one is provided, returned to VSAM, and the processing program issued another GET. (See *z/OS DFSMS Using Data Sets* for information on the EODAD routine.)
8(X'8')	You attempted to store a record with a duplicate key, or there is a duplicate record for an alternate index with the unique key option.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
12(X'C')	An attempt was made to perform sequential or skip-sequential processing against a record whose key/record number does not follow the proper ascending/descending sequential order. The error may occur under any one of the following processing conditions: • For a key-sequenced data set – PUT sequential or skip-sequential processing – GET sequential, single string input only – GET skip-sequential processing and the previous request is not a POINT • For a relative record data set – GET skip-sequential processing – PUT skip-sequential processing
16(X'10')	Record not found, or the RBA is not found in the buffer pool. (If multiple RPL requests are issued for alternate indexes, getting return code 16(X'10') might mean a temporary situation where processing has not been completed on either the base cluster or the associated alternate indexes.)

20(X'14')	Control interval exclusive use conflict. The address of the RPL that owns the resource is placed in the first word in the RPL error message area.
	For VSAM RLS and DFSMStvs, another RPL that is used by this LUWID or UR holds an exclusive lock on this record. This code means that there was an intra-LUWID exclusive control conflict. If an RPL message area of sufficient length is specified, the following information is returned.
	`Offset Length Description`
	
	`0 4 Address of RPL in exclusive control`
	`4 1 Flag Byte: - Not used For RLS`
	`X'00'--neither RPL doing a control area split`
	`X'01'--current RPL doing a control area split`
	`X'02'--other RPL doing a control area split`
	If this request's RPL specifies a MSGAREA of length 4 bytes or greater, the address of an RPL whose lock on this record caused this request to be rejected is returned in the first 4 bytes of MSGAREA. The application may choose to issue an ENDREQ on that RPL and then reissue this POINT, GET NUP, or GET UPD request.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning

21(X'15')	For VSAM RLS and DFSMStvs, another LUWID holds an exclusive lock on this record. The combination of one or more LUWIDs waiting for other record locks held by this LUWID and this LUWID waiting for this record lock produced a deadlock.
	
	If an RPL message area of sufficient length (four bytes or longer) is specified, and the requestor is a commit protocol application (for example, CICS), the following information is returned in the RPL message area:
	
	`Offset Length Description`
	
	`0 4 Address of problem determination area`
	`If you see this error, you are required`
	`to free this area, for example, with:`
	`?STORAGE (RELEASE)`
	`where LENGTH=VPDISIZE, SP=0,KEY=user's key.`

22(X'16')	For VSAM RLS and DFSMStvs, another LUWID holds an exclusive lock on this record. This request waited for the record lock until the timeout interval expired.
	
	If an RPL message area of sufficient length (four bytes or longer) is specified, and the requestor is a commit protocol application (for example, CICS), the following information is returned in the RPL message area:
	
	`Offset Length Description`
	
	`0 4 Address of problem determination area.`
	`If you see this error, you are required`
	`to free this area, for example, with:`
	`?STORAGE (RELEASE)`
	`where LENGTH=VPDISIZE, SP=0,KEY=user's key.`

24(X'18')	Record resides on a volume that cannot be mounted.
	
	For VSAM RLS and DFSMStvs, another LUWID holds a retained lock on this record.
	
	If an RPL message area of sufficient length (four bytes or longer) is specified, the following information is returned in the RPL message area:
	
	`Offset Length Description`
	
	`0 4 Address of problem determination area.`
	`If you see this error, you are required`
	`to free this area, for example, with:`
	`?STORAGE (RELEASE)`
	`where LENGTH=VPDISIZE, SP=0,KEY=user's key.`
	
	For non-RLS, message area information is not returned.

28(X'1C')	Data set cannot be extended because VSAM cannot allocate additional direct access storage space. Either there is not enough space left to make the secondary allocation request, or you attempted to increase the size of a data set while processing with SHAREOPTIONS=4 and DISP=SHR. For VSAM RLS and DFSMStvs, the error can occur for a GET request when the same error has been issued for a preceding PUT request on the same ACB.
32(X'20')	You specified an RBA that does not give the address of any data record in the data set.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
36(X'24')	Key ranges were specified for the data set when it was defined, but no range was specified that includes the record to be inserted.
40(X'28')	Insufficient virtual storage in your address space to complete the request.
44(X'2C')	Work area not large enough for the data record or for the buffer (GET with OPTCD=MVE).
48(X'30')	Invalid options, data set attributes, or processing conditions: • CNV processing • The specified RPL is asynchronous • Chained RPLs • Path processing • Shared resources (LSR/GSR) indeterminate buffer status • Load mode • Fixed-length relative record data set • Data set contains spanned records • User not in key 0 and supervisor state • End-of-volume in process (secondary allocation)
52(X'34')	Invalid options, data set attributes, or processing conditions specified by MVS/DFP. (See X'34' for a list of the invalid options).
54(X'36')	CA Reclaim or CA Reclaim Recovery processing encountered an error.
56(X'38')	Error from catalog update at the beginning of a CI/CA split for backup while open. For VSAM RLS and DFSMStvs, this error indicates an invalid reuse of an RLS RPL. This RPL has position established for VSAM RLS and DFSMStvs access to a data set. The application has changed the ACB or the LUWID, or both. VSAM RLS and DFSMStvs do not permit this form of RPL reuse. This error does not change or lose the string's position. Before changing the ACB or LUWID, the application must issue an ENDREQ on the RPL to release the string's position. RPL reuse violation. The RPL request had positioning information from a previous request and the ACB or LUWID specified in the RPL, or both, did not match that of the prior request.
64(X'40')	There is insufficient storage available to add another string dynamically. Or, the maximum number of place holders that can be allocated to the request has been allocated, and a place holder is not available. For VSAM RLS and DFSMStvs, the limit of 1024 outstanding requests for this ACB has been exceeded.
68(X'44')	The application attempted to use a type of processing (output or control interval processing) that was not specified when the data set was opened.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
72(X'48')	A request was issued in one of the following situations: • The application made a keyed request for access to an entry-sequenced data set. • The application issued a GETIX or PUTIX to an entry-sequenced data set or fixed-length RRDS. For VSAM RLS and DFSMStvs, the application issued a GETIX or PUTIX. GETIX and PUTIX are not supported by VSAM RLS and DFSMStvs.
76(X'4C')	The application issued an addressed or control interval PUT to add to a key-sequenced data set or variable-length RRDS. Or, the application issued a control interval PUT to a fixed-length RRDS.
80(X'50')	The application issued an ERASE request in one of the following situations: • For access to an entry-sequenced data set • For access to an entry-sequenced data set via a path • With control interval access
84(X'54')	The application specified OPTCD=LOC in one of the following situations: • For a PUT request • In the previous request parameter list in a chain of request parameter lists • For UBF processing
88(X'58')	The application issued a sequential GET request without being positioned to it. Or, the application changed from addressed access to keyed access without being positioned for keyed-sequential retrieval. There was no positioning established for sequential PUT insert for a RRDS. Or, the application attempted an illegal switch between forward and backward processing.
92(X'5C')	The application issued a PUT for update, an ERASE without a previous GET for update, or a PUTIX without a previous GETIX.
96(X'60')	The application attempted to change the prime key or key of reference while making an update. Or, for MACRF=RLS, the PUT NUP request attempted to change the key that a prior IDALKADD request specified.
97(X'61')	One of the following situations occurred for a request issued only against a VSAMDB (created by DEFINE CLUSTER DATABASE): • For a POINT or GET DIR request issued against a VSAMDB, neither KEYNAME nor KEYNAMEU was specified in the DEFINE CLUSTER. • For a POINT or GET DIR request, the RPL ARG was not specified, or DBARGLN was not specified. • For a POINT or GET DIR request, the RPL ARG argument could not be converted to an internal key. • For a PUT request, a primary key value found in the document to be inserted or updated could not be converted to an internal key. • For a PUT request, the key name was not found in the document to be inserted or updated.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
98(X'62')	For retrieval, when accessing a VSAMDB (KSDS defined with DATABASE option), an incomplete blocked document was returned. This condition may have occurred during a system interruption of inserting a large VSAMDB document. Use the ERASE command to delete the invalid document.
100(X'64')	The application attempted to change the length of a record while making an addressed update.
104(X'68')	The RPL options are either invalid or conflicting in one of the following ways: • SKP was specified and either KEY was not specified or BWD was specified. • XRBA was not specified in the RPL OPTCD when a GET DIR or a POINT request was issued in ADR or CNV mode with LRD=OFF, and RPLARG points to a nonzero argument (RBA), while processing an extended-addressing data set. • BWD was specified for CNV processing. • FWD and LRD were specified. • Neither ADR, CNV, nor KEY was specified in the RPL. • BFRNO is invalid (less than 1 or greater than the number of buffers in the pool). • WRTBFR, MRKBFR, or SCHBFR was issued, but either TRANSID was greater than 31 or the shared resource option was not specified. • ICI processing was specified, but a request other than a GET or a PUT was issued. • MRKBFR MARK=OUT or MARK=RLS was issued but the RPL did not have a data buffer associated with it. • The RPL specified WAITX, but the ACB did not specify LSR or GSR. • CNV processing is not allowed for compressed data sets. Only VERIFY and VERIFY REFRESH are allowed. • VERIFY was specified for a z/OS UNIX file. • BWD or UPD was specified for a z/OS UNIX file. • DIR was specified for a z/OS UNIX file, that is an FIFO or character special file. • Non-key access issued against extended-format, extended-addressing data set when RBA/XRBA is required for positioning.
106(X'6A')	An invalid internal control block has been detected. A diagnostic dump is taken and the write to DASD failed to prevent possible corruption of the data set.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
108(X'6C')	Incorrect RECLEN. Some possible reasons are: 1. RECLEN specified was larger than the maximum allowed, equal to 0, or smaller than the sum of the length and the displacement of the key field. 2. RECLEN was not equal to record (slot) size specified for a fixed-length RRDS. 3. RECLEN was not sufficient to contain the new alternate index key pointer. With non-unique UPGRADE AIX®'s, the record is automatically increased in size each time a record is added to the base cluster and this can cause an incorrect RECLEN. Make sure the maximum RECORDSIZE on the alternate index is large enough for all base pointers it must contain. 4. RECLEN for a spanned record was longer than 255 segments. Increasing the data CI size to 2560 bytes or more may prevent the error.
109(X'6D')	The index data trap hit, indicating the data set may be already corrupted. No more requests are allowed against the control block structure for the data set. The user is to submit: • The dump • An examine output after closing the data set • Prints of the index and data components as soon as the error is returned. More documents may be required, depending on what is known about the problem. Close the data set, restore it, and re-open to re-access if necessary.
112(X'70')	KEYLEN specified was too large or equal to 0.
116(X'74')	During initial data set loading (that is, when records are being stored in the data set the first time it is opened), GET, POINT, ERASE, direct PUT, skip-sequential PUT, or PUT with OPTCD=UPD is not allowed. For initial loading of a fixed length RRDS, the request was other than a PUT insert.
120(X'78')	Request was operating under an incorrect TCB. For example, an end-of-volume call or a GETMAIN macro was necessary to complete the request, but the request was issued from a task other than the one that opened the data set. The request can be resubmitted from the correct task if the new request reestablishes positioning.
124(X'7C')	A request was cancelled for a user JRNAD exit.
128(X'80')	A loop exists in the index horizontal pointer chain during index search processing.
132(X'84')	An attempt was made in locate mode to retrieve a spanned record.
136(X'88')	The application attempted an addressed GET of a spanned record in a key-sequenced data set.
140(X'8C')	The spanned record segment update number is inconsistent.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
144(X'90')	Invalid pointer (no associated base record) in an alternate index. If multiple RPL requests are issued for alternate indexes, getting return code 144(X'90') might mean a temporary situation where processing has not been completed on either the base cluster or the associated alternate indexes. For example, you have issued multiple RPL requests including erase requests to the path or base cluster, and got a return code of X'90'. This might be a temporary situation where the base cluster has been erased, but the associated alternate index has not been erased. If you provide a message area using the MSGAREA parameter of the RPL macro, VSAM returns the address of an RPL doing the erase when the return code X'90' was set.
148(X'94')	The maximum number of pointers in the alternate index has been exceeded.
152(X'98')	• For LSR, not enough buffers are available to process the request and more could not be added dynamically. • For GSR, not enough buffers are available to process the request • For RLS and DFSMStvs, the dataspace buffer pool was exhausted.
156(X'9C')	Invalid control interval detected during keyed processing, an addressed GET UPD request failed because control interval flag was on, or an invalid control interval or index record was detected. The RPL contains the invalid control interval's RBA.
158(X'9E')	For RLS, this system has found the data set corrupted and cannot access it. Please close the data set, restore it, and re-open to re-access it if necessary.
160(X'A0')	One or more candidates were found that have a modified buffer marked to be written. The buffer was left in write status with valid contents. With this condition, it is possible to have other buffers invalidated or found under exclusive control.
165(X'A5')	For RLS and DFSMStvs, either the IDARECOV request was specified as TYPE=LL and the sphere was not in lost locks state for this subsystem, or TYPE=NONRLS was specified and the sphere was not in NONRLSUPDATE permitted state.
167(X'A7')	The field QUIESTYP in the IFGQUIES parameter area specifies an invalid request type or the eye-catcher in IFGQUIES is invalid. For RLS, IDAQUIES type QUIBWO, QUICOPY, caller is not in supervisor state. Invalid Quiesce request.
168(X'A8')	For MACRF=RLS, the pointer in the RPL to the record is zero.
169(X'A9')	For RLS and DFSMStvs, the IDAQUIES, IDARETLK TYPE=SS, DARECOV TYPE=LL, or IDARECOV TYPE=NONRLS request failed because the Catalog Locate command issued for the specified sphere or component name failed.
170(X'AA')	The QUIOPEN, QUICEND, or QUIBEND request is rejected because the requested unquiesce operation is already started for this sphere.
172(X'AC')	For RLS and DFSMStvs, the IDAQUIES, IDARETLK TYPE=SS, IDARECOV TYPE=LL, or IDARECOV TYPE=NONRLS request failed because the specified sphere is not an SMS VSAM data set.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
176(X'B0')	For RLS and DFSMStvs, the shared-latch obtain failed for the record management request. Or the ACB specified in the Record Management request or in the IDARETLK TYPE=SS, IDARECOV TYPE=LL, or IDARECOV TYPE=NONRLS request is not a valid ACB open for RLS or DFSMStvs.
180(X'B4')	For MACRF=RLS, an invalid request for a nonrecoverable data set.
181(X'B5')	This IDAQUIES request is rejected because the requestor does not have update authority to the sphere. Issued for the following: * This is a type QUICLOSE request. Successful completion of the request results in a catalog update to mark the sphere quiesced. Because the requestor does not have update authority, the request is rejected. The catalog shows this sphere is quiesced. Successful completion of the QUIOPEN request would result in an update to the catalog to reset the quiesced state of the sphere. because the requestor does not have update authority, the request is rejected.

182(X'B6')	For rls, the IDAQUIES request is rejected because an IDAQUIES is already in progress for this sphere.
	
	If an RPL message area (address in RPLERMSA) of sufficient length (specified in RPLEMLEN) is specified, the following information is returned:
	
	`Offset  Length Description`
	`------  ------ -----------`
	`0         1    Type of quiesce event already in progress for this sphere.`
	`              Quiesce type constants are defined in IFGQUIES mac.`

183(X'B7')	IDAQUIES request rejected because data set is migrated.
184(X'B8')	For MACRF=RLS, the application issued an ABEND condition while VSAM was processing this request. The VSAM RLS FRR (Functional Recovery Routine) intercepted the failure and failed the VSAM request with this reason code.
185(X'B9')	For MACRF=RLS, the user task was cancelled while the request was being processed.
186(X'BA')	For MACRF=RLS, an abend occurred in an attempt to access user storage during logging. This might have happened if the application program issued FREEMAIN or STORAGE RELEASE for the buffers. For base VSAM, an EOV initialize failure occurred. For example, with data set name sharing, if dynamic allocation (DYNALLOC) is issued before the first OPEN of the data set and that first ACB is later closed, EOVs against the data set will fail because EOV is unable to find the TIOT offset indicated in the shared control block structure. In that case, either close all DDNAMEs for the data set and then do an OPEN, or ensure that the first DDNAME opened remains open.
187(X'BB')	For MACRF=RLS, an error occurred with partial EOV processing. For Base VSAM, failed to setup a ESTAE recovery routine.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
188(X'BC')	For MACRF=RLS, the sphere is in lost locks state. A record management request was issued by this SUBSYSNM, but these requests are not allowed until the sphere is out of lost locks state.
189(X'BD')	For MACRF=RLS, a lock for the VSAM request required space in the record table, but the table was full. Installation action is needed to modify the CFRM policy and rebuild the lock structure.
190(X'BE')	Partial EOV error.
192(X'C0')	Invalid relative record number.
193(X'C1')	An RLS request failed during a read I/O and a dump was generated without terminating the VSAM server address space.
196(X'C4')	The application issued an addressed request to a fixed- or variable-length RRDS.
200(X'C8')	The application attempted addressed or control interval access through a path.
201(X'C9')	For RLS or DFSMStvs, the IDARETLK TYPE=SS request failed because the specified data set does not exist.
204(X'CC')	PUT insert requests (or for VSAM RLS or DFSMStvs, IDALKADD requests) are not allowed in backward mode.
205(X'CD')	For LSF, indicates invalid CONTOKEN.
206(X'CE')	For DFSMStvs, indicates that the request was rejected because the data set is quiesced or quiescing for copy. Wait for the copy to complete and then retry the request. For NSR, LSR, or GSR, this reason code indicates a validity check error for shareoptions 3,4.
207(X'CF')	Indicates that DFSMStvs processing is currently unavailable because DFSMStvs is quiescing or disabling. Close all data sets to allow the quiesce/disable process to complete.
208(X'D0')	An ENDREQ was issued against an RPL that has an outstanding WAIT against its associated ACB. No ENDREQ processing was done.
211(X'D3')	For DFSMStvs, this indicates that the forward recovery log is unusable for this system as a result of either a failure by OPEN to complete connect processing to the logstream, or an error occurred while writing to this logstream. See accompanying DFSMStvs logger messages for appropriate action. For LSR, the cache request is purged.
212(X'D4')	During control area split processing, an existing condition prevents the split of the index record. Redefine the cluster and increase the index CI size. See *z/OS DFSMS Using Data Sets* to determine how to estimate the effective CI size of index component.
213(X'D5')	For DFSMStvs, indicates that the undo log is unavailable for processing. For LSR, no connectivity to the cache structure.
215(X'D7')	An update to the data set defined with LOGREPLICATE failed because it violates the condition that all log replicate eligible data sets updated in a unit of recovery (UR) need to use the same logstreamid.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
216(X'D8')	For MACRF=RLS, LUWID specified in the RPL does not exist for the subsystem name specified in the ACB.
218(X'DA')	Unrecognizable return code from SVC109.
220(X'DC')	DFSMStvs was unable to complete the request because RRS is currently unavailable.
224(X'E0')	MRKBFR OUT was issued for a buffer with invalid contents.
228(X'E4')	Caller in cross-memory or SRB mode is not in supervisor state, or RPL of caller in SRB or cross-memory mode specifies ASY. For MACRF=RLS, the caller is not in primary ASC mode, or the caller issued a record management request with an FRR in effect, or the task that opened the ACB is not in the caller's task hierarchy.
229(X'E5')	The record length changed during decompression processing.
230(X'E6')	The processing environment was changed by the user of the UPAD exit.
232(X'E8')	UPAD error; ECB was not posted by user in cross-memory mode.
235(X'EB')	VSAM RLS or DFSMStvs internal error.
236(X'EC')	Validity check error for SHAREOPTIONS 3 or 4.
237(X'ED')	Reserved.
238(X'EE')	Reserved.
239(X'EF')	Reserved.
240(X'F0')	For shared resources, one of the following is being performed: (1)an attempt is being made to obtain a buffer in exclusive control, (2)a buffer is being invalidated, or (3)the buffer use chain is changing. For more detailed feedback, reissue the request.
241(X'F1')	Reserved.
242(X'F2')	Reserved.
243(X'F3')	Reserved.
244(X'F4')	Register 14 stack size is not large enough.
245(X'F5')	Severe error returned by compression management services during a compress call. Additional problem determination is provided in the RPL message area.
246(X'F6')	An error occurred during an expansion of the user record for an extended-function data set. The RPL message area contains additional problem determination.
248(X'F8')	Register 14 return offset went negative.
249(X'F9')	For DFSMStvs, indicates that undo logging failed because the record length is greater than the installation-defined maximum for the log. For LSR XI, invalid vector token.
250(X'FA')	No valid dictionary token exists for the data set. VSAM is unable to decompress the data record.

Table 17. Logical Error Reason Codes in the Feedback Area of the Request Parameter List (continued)

Reason Code (RPLERRCD) When Register 15=8(X'8')	Meaning
251(X'FB')	Internal VSAM RLS error.
252(X'FC')	Record mode processing is not allowed for a linear data set.
253(X'FD')	VERIFY is not a valid function for a linear data set.

When the search argument You supply for a POINT or GET request is greater than the highest key in the data set, the reason code in the feedback area depends on the RPL's OPTCD values, as shown in the following table:

Request Type	RPLs OPTCD Options	Reason Code When Register 15=8(X'8')
POINT	GEN,KEQ	16(X'10')
POINT	GEN,KGE	4(X'4')
POINT	FKS,KEQ	16(X'10')
POINT	FKS,KGE	4(X'4')
GET	GEN,KEQ,DIR	16(X'10')
GET	GEN,KGE,DIR	16(X'10')
GET	FKS,KEQ,DIR	16(X'10')
GET	FKS,KGE,DIR	16(X'10')
GET	GEN,KEQ,SKP	16(X'10')
GET	GEN,KGE,SKP	4(X'4')
GET	FKS,KEQ,SKP	16(X'10')
GET	FKS,KGE,SKP	4(X'4')

Positioning following logical errors

VSAM is unable to maintain positioning after every logical error. Whenever positioning is not maintained following an error request, You must reestablish it before processing resumes.

Positioning may be in one of four states following a POINT or a direct request that found a logical error:

Yes
VSAM is positioned at the position in effect before the request in error was issued.

No
VSAM is not positioned, because no positioning was established at the time the request in error was issued.

New
VSAM is positioned at a new position.

U
VSAM is positioned at an unpredictable position.

N/A
The reason code is not applicable to the type of processing indicated.

Table 18 on page 136 shows which positioning state applies to each reason code listed for sequential, direct, and skip-sequential processing. "N/A" indicates the reason code is not applicable to the type of processing indicated.

Table 18. Positioning States of Reason Codes Listed for Sequential, Direct, and Skip-Sequential Processing

Reason Code (RPLERRCD) When Register 15=8(8)	Sequential	Direct	Skip-Sequential
4 (X'4')	Yes	No	Yes
8 (X'8')"1" on page 138	Yes	No	New
12 (X'C')	Yes	N/A	Yes
16 (X'10')	No	No	No
20 (X'14')	U	No"2" on page 138	No"2" on page 138
21 (X'15')	Yes"3" on page 138	New	New
22 (X'16')	Yes"3" on page 138	New	New
24 (X'18')	Yes"3" on page 138	No	No
28 (X'1C')	Yes	No	Yes
32 (X'20')	No	No	N/A
36 (X'24')	Yes	No	New
40 (X'28')	Yes	No	No
44 (X'2C')	Yes	New	Yes
48 (X'30')	U	U	U
52 (X'34')	U	U	U
56 (X'38')	Yes	Yes	Yes
64 (X'40')	No	No	No
68 (X'44')	Yes	Yes	Yes
72 (X'48')	Yes	Yes	Yes
76 (X'4C')	Yes	Yes	Yes
80 (X'50')	Yes	Yes	Yes
84 (X'54')	Yes	Yes	Yes
88 (X'58')	Yes	Yes	Yes
92 (X'5C')	Yes	Yes	Yes
96 (X'60')	Yes	Yes	Yes
100 (X'64')	Yes	Yes	Yes
104 (X'68')	Yes	New	Yes
108 (X'6C')	Yes	New	Yes
112 (X'70')	Yes	Yes	Yes
116 (X'74')	Yes	Yes	Yes
120 (X'78')	Yes	No	No
124 (X'7C')	No	No	No
128 (X'80')	Yes	No	No

Table 18. Positioning States of Reason Codes Listed for Sequential, Direct, and Skip-Sequential Processing (continued)

Reason Code (RPLERRCD) When Register 15=8(8)	Sequential	Direct	Skip-Sequential
132 (X'84')	Yes	New	Yes
136 (X'88')	No	No	N/A
140 (X'8C')	Yes	New	Yes
144 (X'90')	Yes	Yes	Yes
148 (X'94')	Yes	Yes	Yes
152 (X'98')	Yes	No	No
156 (X'9C')	Yes	No	No
160 (X'A0')	N/A	No	N/A
168 (X'A8')	N/A	N/A	N/A
169 (X'A9')	N/A	N/A	N/A
172 (X'AC')	N/A	N/A	N/A
176 (X'B0')	N/A	N/A	N/A
180 (X'B4')	Yes	Yes	Yes
181 (X'B5')	N/A	N/A	N/A
182 (X'B6')	N/A	N/A	N/A
184 (X'B8')	U	U	U
186 (X'BA')	Yes	Yes	Yes
190 (X'BE')	Yes"3" on page 138	No	Yes
192 (X'C0')	Yes	Yes	Yes
196 (X'C4')	Yes	Yes	Yes
200 (X'C8')	Yes	Yes	Yes
201 (X'C9')	N/A	N/A	N/A
204 (X'CC')	Yes	Yes	Yes
208 (X'D0')	Yes	Yes	Yes
211 (X'D3')	No	No	No
212 (X'D4')	U	U	U
216 (X'D8')	N/A	N/A	N/A
224 (X'E0')	N/A	No	N/A
228 (X'E4')	No	No	No
229 (X'E5')	New	New	New
230 (X'E6')	Yes	Yes	Yes
232 (X'E8')	No	No	No

Table 18. Positioning States of Reason Codes Listed for Sequential, Direct, and Skip-Sequential Processing (continued)

Reason Code (RPLERRCD) When Register 15=8(8)	Sequential	Direct	Skip-Sequential
235 (X'EB')	U	U	U
236 (X'EC')	Yes	Yes	Yes
237 (X'ED')	U	U	U
238 (X'EE')	U	U	U
239 (X'EF')	U	U	U
240 (X'F0')	Yes	Yes	Yes
241 (X'F1')	No	No	No
242 (X'F2')	U	U	U
243 (X'F3')	No	No	No
244 (X'F4')	U	U	U
245 (X'F5')	New	New	New
246 (X'F6')	New	New	New
248 (X'F8')	U	U	U
249 (X'F9')	Yes	Yes	Yes
250 (X'FA')	New	New	New
251 (X'FB')	U	U	U
252 (X'FC')	No	No	No
253 (X'FD')	No	No	No

Notes:

1. A subsequent GET SEQ will retrieve the duplicate record. However, a subsequent GET SKP for the same key will get a sequence error. In a fixed- or variable-length RRDS, a subsequent PUT SEQ positions to the next slot (whether the slot is empty or not).
2. For NSR requests, PUT UPD, DIR or UPD, SKP retains positioning. The RPL contains an RBA that could not be obtained for exclusive control.
3. For MACRF=RLS, position will advance to next record on next request.

Reason code (physical errors)

If a physical error occurs and you have no SYNAD routine (or the SYNAD exit is inactive), VSAM returns control to your program following the last executed instruction. The return code in register 15 indicates a physical error (12). The RPL feedback area contains a reason code identifying the error. The RPL message area contains more details about the error. Register 1 points to the request parameter list. The RBA field in the request parameter list gives the relative byte address of the control interval in which the physical error occurred. Table 19 on page 139 gives the reason codes in the feedback area and explains what each indicates.

Table 19. Physical Error Reason Codes in the Feedback Area of the Request Parameter List

Reason Code (RPLERRCD) When Register 15=12(X'0C')	Meaning
4(X'4')	Read error occurred for a data set.
8(X'8')	Read error occurred for an index set.
12(X'C')	Read error occurred for a sequence set.
16(X'10')	Write error occurred for a data set.
20(X'14')	Write error occurred for an index set.
24(X'18')	Write error occurred for a sequence set.
36(X'24')	For MACRF=RLS, a CF cache structure connectivity failure occurred.
40(X'28')	For MACRF=RLS, a CF cache structure failure occurred.
44(X'2C')	For extended format data sets, the suffix for a physical record in the CI at the RBA specified in the RPL is invalid.

Table 20 on page 139 shows the format of a physical error message. The format and some of the contents of the message are purposely similar to the format and contents of the SYNADAF message, which *z/OS DFSMS Macro Instructions for Data Sets* describes.

Table 20. Physical Error Message Format for Non-RLS Processing

Field	Bytes	Length	Description
Message Length	0-1	2	Binary value of 128.
	2-3	2	Unused (0)
Message Length-4	4-5	2	Binary value of 124 (provided for compatibility with SYNADAF Message).
	6-7	2	Unused (0)
Address of I/O Buffer	8-11	4	The I/O buffer associated with the data where the error occurred.
The rest of the message is in printable format			
Date	12-16	5	YYDDD (year and day)
	17	1	Comma (,)
Time	18-25	8	HHMMSSTH (hour, minute, second, tenths and hundredths of a second.
	26	1	Comma (,)
RBA	27-38	12	Relative byte address of the record where the error occurred.
	39	1	Comma (,)
Component Type	40	1	"D"(Data) or "I"(Index)
	41	1	Comma (,)
Volume Serial Number	42-47	6	Volume serial number of the volume where the error occurred.
			For z/OS UNIX files, contains "********"

Table 20. Physical Error Message Format for Non-RLS Processing (continued)

Field	Bytes	Length	Description
	48	1	Comma (,)
Job Name	49-56	8	Name of the job where error occurred.
	57	1	Comma (,)
Step Name	58-65	8	Name of the job step where the error occurred.
	66	1	Comma (,)
Unit	67-70	4	The device number where the error occurred.
			For z/OS UNIX files, this field contains "****"
	71	1	Comma (,)
Device Class	72-73	2	The type of device where the error occurred. (Always DA for direct access.)
	74	1	Comma (,)
ddname	75-82	8	The ddname of the DD statement defining the data set where the error occurred.
	83	1	Comma (,)
Channel	84-89	6	The channel command that received the error in the first two bytes, followed by “-OP”
			For z/OS UNIX files, this field contains the request which resulted in the error.
			Either a GET, PUT, CHECK, POINT, or ENDREQ request.
	90	1	Comma (,)

Messages 91–105 are described below.

Message 91-105 15 Messages are divided according to ECB completion codes:

X'41' "INCORR LENGTH"
"UNIT EXCEPTION"
"PROGRAM CHECK"
"PROTECTION CHK"
"CHAN DATA CHK"
"CHAN CTRL CHK"
"INTFCE CTRL CHK"
"CHAINING CHK"
"UNIT CHECK"
"SEEK CHECK"

If the type of unit check can be determined, the "UNIT CHECK" message is replaced by one of the following:

"CMD REJECT"
"INT REQ"
"BUS OUT CK"
"EQP CHECK"
"DATA CHECK"
"OVER RUN"

"TRACK COND CK"
"COUNT DATA CHK"
"TRACK FORMAT"
"CYLINDER END"
"NO RECORD FOUND"
"FILE PROTECT"
"MISSING A.M."
"OVERFL INCP"
X'48' "PURGED REQUEST"
X'4A' "I/O PREVENTED"
X'4F' "R.HA.R0. ERROR"
"INVALID SUFFIX"

Table 21 on page 141 and Table 22 on page 141 show information about messages 106–127.

Table 21. Physical Error Message Format for any other ECB completion code

Field	Bytes	Length	Description
			"UNKNOWN COND".
			For z/OS UNIX files, this field contains the service which encountered an error, in the form "OMVS-*nnnnnnnn*" where *nnnnnnnn* is the name of the service.
	106	1	Comma (,)
Physical Direct	107-120	14	BBCCHHR (bin, cylinder, head, and record)
Access Address			For z/OS UNIX files, this field contains the return and reason code from the failing service in the form "*xxxx-yyyyyyyy*" consisting of a 2-byte hexadecimal return code and a 4-byte hexadecimal reason code.
	121	1	Comma (,)
Access Method	122-127	6	"VSAM"
			For z/OS UNIX files, this field contains "VSAM"

Table 22. Physical Error Message Format for CF Failure with VSAM RLS or DFSMStvs Processing

Field	Bytes	Length	Description
Message Length	0-1	2	Binary value of 128
	2-3	2	Unused (0)
Message Length-4	4-5	2	Binary value of 124 (provided for compatibility with SYNADAF Message).
	6-7	2	Unused (0)
Address of I/O Buffer	8-11	4	The I/O buffer associated with the data where the error occurred.
The rest of the message is in printable format			
Date	12-16	5	YYDDD (year and day)
	17	1	Comma (,)
Time	18-25	8	HHMMSSTH (hour, minute, second, tenths and hundredths of a second.

Table 22. Physical Error Message Format for CF Failure with VSAM RLS or DFSMStvs Processing (continued)

Field	Bytes	Length	Description
	26	1	Comma (,)
RBA	27-38	12	Relative byte address of the record where the error occurred.
	39	1	Comma (,)
Component Type	40	1	"D"(Data) or "I"(Index)
	41	1	Comma (,)
Volume Serial	42-47	6	For MACRF=RLS, this field does not apply and is set to asterisks.
	48	1	Comma (,)
Job Name	49-56	8	Name of the job where the error occurred.
	57	1	Comma (,)
Step Name	58-65	8	Name of the job step where the error occurred.
	66	1	Comma (,)
Unit	67-70	4	For MACRF=RLS, this field does not apply and is set to asterisks.
	71	1	Comma (,)
Device Type	72-73	2	For MACRF=RLS, this field is set to "CS" for CF cache structure.
	74	1	Comma (,)
ddname	75-82	8	The ddname of the DD statement defining the data set where the error occurred.
	83	1	Comma (,)
Channel	84-89	6	For MACRF=RLS, this field is set to "CFREAD" or "CFWRT" indicating if the CF operation is a read or write.
	90	1	Comma (,)
Message	91-105	15	For MACRF=RLS, you receive either CF structure failure message or loss of connectivity message.
			"CF STR FAILURE"
			"CF CON FAILURE"
	106	1	Comma (,)
Physical Direct Access Address	107-120	14	14-character cache structure name.
	121	1	Comma (,)
Access Method	122-127	6	"VSAM"

Reason code (server errors)

If a server failure occurs in the SMSVSAM address space for VSAM RLS, the return code in register 15 indicates a server error (16). The RPL feedback area contains a reason code identifying the type of server failure. Table 23 on page 143 gives the reason codes in the feedback area and explains the associated failures.

Table 23. Server Failure Reason Codes in the Feedback Area of the Request Parameter List

Return Code (RPLERRCD) When Register 15=16(X'10')	Meaning
4(X'4')	VSAM server address space is detected to be inactive, uninitialized, or at a different server instance.
8(X'8')	Server is terminating; CF connection is lost.

Return codes from macros used to share resources among data sets

VSAM has a set of macros that allow you to share I/O buffers, I/O related control blocks, and channel programs among VSAM data sets.

BLDVRP return codes

VSAM returns a code in register 15 that indicates if the BLDVRP request was successful. Table 24 on page 143 describes these return codes.

Table 24. Return Codes in Register 15 After BLDVRP Request

Return Code	Meaning
0(X'0')	VSAM completed the request.
4(X'4')	The requested data resource pool or index resource pool already exists in the address space (LSR) or in the system protect key (GSR).
8(X'8')	Insufficient virtual storage space to satisfy request. GETMAIN or ESTAE failed.
12(X'C')	Opens have already been issued against the shared buffer pool BLDVRP is building. **Rule:** As a VSAM user, you are responsible for ensuring that the BLDVRP/DLVRP requests are serialized with the open or close requests. VSAM cannot completely detect the lack of such serialization.
16(X'10')	TYPE=GSR is specified but the program that issued BLDVRP is not in supervisor state with protection key 0 to 7.
20(X'14')	STRNO is less than 1 or greater than 255, or parameters are invalid.
24(X'18')	BUFFERS is specified incorrectly. A size or number is invalid.
32(X'20')	The resource pool already exists above 16 megabytes and the request was for storage below 16 megabytes. Or, the resource pool already exists below 16 megabytes and the request was for storage above 16 megabytes.
36(X'24')	BLDVRP was issued to build an index resource pool but the required corresponding data resource pool does not exist.
40(X'28')	The size for Hiperspace buffers is specified incorrectly. The buffer size must be a multiple of 4K with a maximum size of 32K.

Table 24. Return Codes in Register 15 After BLDVRP Request (continued)

Return Code	Meaning
44(X'2C')	Attention: At least one request for Hiperspace buffers was rejected because of insufficient expanded storage. The specific buffer subpools rejected may be located by checking for the BLPBFNHS indicator in the Hiperspace buffer request list. The BLDVRP request was otherwise successful. This return code is also valid for jobs indicating RESTART processing.
45(X'2D')	Attention: All hiperspace creates have failed because no expanded storage was installed on the system. BLDVRP processing continued as if no hiperspace buffers were requested. The BLDVRP request was otherwise successful. This return code is also valid for jobs indicating RESTART processing.
48(X'30')	A buffer size specified for a Hiperspace buffer pool is not equal to any of the buffer sizes specified for the virtual buffer pool.
52(X'34')	Another BLDVRP or DLVRP on the same shared pool is in progress.

DLVRP return codes

VSAM returns a code in register 15 that indicates if the DLVRP request was successful. Table 25 on page 144 describes these return codes.

Table 25. Return Codes in Register 15 Following DLVRP Request

Return Code	Meaning
0(X'0')	VSAM completed the request.
4(X'4')	There is no resource pool to delete.
8(X'8')	Insufficient virtual storage space to satisfy request. GETMAIN or ESTAE failed.
12(X'C')	There is at least one open data set using the resource pool.
16(X'10')	TYPE=GSR is specified, but the program that issued DLVRP is not in supervisor state with protection key 0 to 7.
20(X'14')	Another BLDVRP or DLVRP on the same shared pool is in progress.

End-of-volume return codes

End-of-volume returns a code in register 15 that indicates if the request was successful. Table 26 on page 144 describes these return codes.

Table 26. Return Codes in Register 15 Following End-of-Volume

Return Code	Meaning
0(X'0')	Successful.
4(X'4')	The requested volume could not be mounted.
8(X'8')	The requested amount of space could not be allocated.
12(X'C')	I/O operations were in progress when end-of-volume was requested.
16(X'10')	The catalog could not be updated.

SHOWCAT return codes

VSAM returns a code in register 15 that indicates whether the SHOWCAT request was successful. Table 27 on page 145 describes these return codes.

Table 27. SHOWCAT Return Codes

Return Code	Meaning
0(X'00')	VSAM completed the task.
4(X'04')	The area specified in the AREA operand is too small to display all pairs of fields for the associated objects.
8(X'08')	There is insufficient virtual storage to complete the task. (A GETMAIN failed.)
12(X'0C')	Either the ACB address is invalid, or the VSAM master catalog does not exist, or it is not open.
16(X'10')	The address specified in the AREA operand is outside the partition or address space of the program that issued SHOWCAT.
20(X'14')	The named object or control interval does not exist.
24(X'18')	There was an I/O error in gaining access to the catalog.
28(X'1C')	The control interval number is invalid.
32(X'20')	The catalog record does not describe a C, D, G, I, R, or Y type of object.
36(X'24')	The interrelationship among catalog entries is in error. For example, another type.
40(X'28')	There was an unexpected error code returned from catalog management to the SHOWCAT processor.

Part 2. Non-VSAM macro instructions

Chapter 4. Introduction to non-VSAM programming

Macro instructions described in this section are for the access methods other than VSAM. They are called non-VSAM macros or basic access method (BAM) macros. Use BAM to organize data. Usually the system maintains information about that data in a catalog.

Perform BAM functions using the following methods:

- Access method services. You can define non-VSAM data sets and perform certain other services using a multi-function service program called *access method services* (IDCAMS). Certain access method services commands such as ALLOCATE, ALTER, DEFINE NONVSAM, DELETE, LISTCAT, PRINT, and REPRO are available for use with non-VSAM data sets.
- Job control language. Use JCL to define or allocate to non-VSAM data sets.
- Dynamic allocation. Use dynamic allocation to define or allocate to data sets, which is SVC 99. For more information about dynamic allocation, see *z/OS MVS Programming: Authorized Assembler Services Guide*. BAM supports the XTIOT, NOCAPTURE, and DSAB-above-the-line options of dynamic allocation.

  The non-VSAM macros can generate reenterable code, depending on the form in which parameters are expressed.

BAM macro instructions

The choice of which non-VSAM macro to use depends on which access method is appropriate for the type of data set being processed:

- Basic and queued sequential access method (BSAM and QSAM) macros are used to process sequential data sets on DASD, members of partitioned data sets or PDSEs, z/OS UNIX files, magnetic tape files, subsystem data sets, TSO terminals and unit record devices.
- Basic partitioned access method (BPAM) macros are used to process partitioned data sets and PDSEs.
- Basic direct access method (BDAM) macros are used to process direct data sets.
- Basic and queued indexed sequential access method (BISAM and QISAM) macros were designed to process indexed sequential data sets, which are no longer supported. All indexed sequential data sets should have been converted to key sequenced data sets (KSDS) prior to your migration to z/OS V1R7. However, BISAM and QISAM macro descriptions in this topic were written as if you were accessing real ISAM data sets, and some parts of this topic describe functions that are no longer supported.

  The ISAM interface for VSAM, described in *z/OS DFSMS Using Data Sets*, helps in converting programs from ISAM to VSAM, and the description of that interface may help you understand this topic. To use real indexed sequential data sets, see documentation for z/OS releases prior to z/OS V1R7.

The macros are in the system macro library SYS1.MACLIB.

Programs in PDSEs

You can store executable programs in PDSE libraries. Although structurally identical, PDSE libraries are of two types:

- A *data library*, which contains source programs, user data, and other record-oriented information.
- A *program library*, which contains executable programs referred to as program objects.

The type of library is determined, not at allocation time, but when the first member is stored in it. For more information about program objects and libraries, see *z/OS MVS Program Management: User's Guide and Reference* and *z/OS MVS Program Management: Advanced Facilities*.

Data above the 2 GB bar

There are only two non-VSAM access method macros that accept a 64-bit virtual address. The READ and WRITE macros support the data area address being a 64-bit virtual address if the data set is extended format and not compressed format. Your program must be running in 31-bit mode. You must code the SF64 or SF64P option on the READ or WRITE macro.

If the 64-bit address points above the 2 GB bar, then the storage must satisfy one of these conditions:

- It was obtained by issuing the IARST64 macro with REQUEST=GET. The maximum length of storage that is gotten by this means is 128 KB.
- It was obtained by issuing the IARV64 macro with the REQUEST=GETSTOR and CONTROL=AUTH options. This requires your program to run in supervisor state or system key. The minimum storage length is 1 MB.

The following example code shows these conditions.

```
GREATAPP START
GREATAPP RMODE ANY              Can reside above the line
GREATAPP AMODE 31
     . . .
         LHI   R0,LenDECB+8     Get space for DECB plus 64-bit pointer
         GETMAIN R,LV=(0),LOC=24  Get storage for DECB below the line
         LR    R2,R1            Save DECB address
         MVC   0(LenDECB,R2),MyDECB  Copy base DECB to dynamic storage
         LH    R0,DCBBLKSI      Get maximum block length
         ST    R0,TempWord
         SAM64                  Switch to 64-bit to get storage
         SYSSTATE AMODE64=YES
         IARST64 REQUEST=GET,SIZE=TempWord,COMMON=NO,                  *
               OWNINGTASK=CURRENT,FPROT=YES,TYPE=PAGEABLE,             *
               CALLERKEY=YES,FAILMODE=ABEND,REGS=SAVE   Get data area
         SAM31
         SYSSTATE AMODE64=NO
         LGR   R3,R1            Save address of area above the bar
         STG   R3,LenDECB(,R2)  Set address of data area
         READ  (R2),SF64P,MyDCB,LenDECB(,R2),MF=E
         CHECK (R2)             Await I/O completion
         SAM64                  Switch to 64-bit to see data
         SYSTATE AMODE64=YES
     . . .
         READ  MYDECB,SF,MF=L   (Same as SF64)
LenDECB  EQU   *-MYDECB         Length of my DECB
TempWord DS    F
```

Note: The example code must save and restore all 8 bytes of registers 2 - 14 even though it might not be entered in 64-bit mode. You can use the STMH(Store Multiple High) instruction to save the contents of those registers in the work area of this module, as the caller's register save area might be too small.

Data above the 16MB line

The BSAM, QSAM, and BPAM access methods allow data areas to be located above the 16MB line. This support includes allowing the caller to issue most SAM, and PAM macros in 31-bit addressing mode regardless of whether the data is above or below the 16MB line.

The support for areas above the line is provided for the following devices:

- DASD, including UNIX files
- Tape
- Subsystem (for example, spooled)
- Dummy
- VIO
- Unit record

The support for areas above the line is not provided for the following devices:

- TSO terminal
- OCR/MICR 3886, 3890, 1287, 1288. These devices are for optical character recognition and magnetic ink character recognition.

For the above devices, issue macros other than OPEN and CLOSE in 24-bit addressing mode. In the current release the DCBE has no effect.

To take advantage of providing data areas above the 16 MB line for BSAM, and BPAM macros, the issuer of READ, WRITE, and CHECK must execute in 31-bit addressing mode.

To take advantage of providing data areas above the 16MB line for QSAM macros, the issuer of GET, PUT, and PUTX must execute in 31-bit addressing mode. To take advantage of QSAM buffers above the line, the user must tell OPEN to obtain the buffers above the line via the DCBE macro and the issuer of GET, PUT, and PUTX must then execute in 31-bit addressing mode.

If the issuer of READ, WRITE, CHECK, GET, PUT, and PUTX executes in 31-bit addressing mode, then all of the following must have 31-bit addresses and can reside above or below the 16MB line:

- Data address in the DECB (BSAM) or in the GET or PUT macro (QSAM move mode).
- Save area in register 13.
- DCB extension (DCBE).
- QSAM buffers obtained at OPEN where the DCBE is present and the user has coded RMODE31=BUFF on the DCBE macro indicating that OPEN can get buffers above the 16MB line (QSAM).
- EODAD address specified in the DCBE (DCBE EODAD=addr) (BSAM, QSAM, and BPAM).
- SYNAD address specified in the DCBE (DCBE SYNAD=addr) (BSAM, QSAM, and BPAM). In case your routine uses register 15 as a base register, the SYNADAF macro modifies the high order byte.
- Key address in the DECB.
- Area containing block address (RBA, TTR, or MBBCCHHR) in the DECB.

If the issuer of an access method macro executes in 31-bit addressing mode, the following must have valid 31-bit addresses but must reside below the 16 MB line:

- DECB (BSAM).
- DCB address on any macro (including the DECB) or in a register.
- BSAM or BPAM buffers obtained by OPEN (BSAM). OPEN obtains BSAM or BPAM buffers only when you code BUFNO on the DCB macro or the DD statement.

In any addressing mode, the following can reside above the 16 MB line if you have a way to set it:

- OPEN and CLOSE parameter list when you code MODE=31.
- DCBE, except with the FIND macro.

The following must reside below the line because the addresses are only three bytes:

- OPEN and CLOSE parameter list when MODE=24 is coded or defaulted.
- DCB exit list.
- Routines and areas pointed to by the exit list. All exit list exit routines are entered in 24-bit addressing mode.
- EODAD address in the DCB. The user's EODAD routine will be entered in the addressing mode of the issuer of the CHECK, GET, or FEOV.
- SYNAD address in the DCB. The user's SYNAD routine will be entered in the addressing mode of the issuer of the CHECK, GET, or PUT. In case your routine uses register 15 as a base register, the SYNADAF macro modifies the high order byte.
- Area containing next block address in the DECB.
- SETPRT parameter list and areas that it points to.
- STOW parameter list.

Following is a complete list of SAM macros which do not support buffers which reside above the line:

- BUILD
- BUILDRCD
- FREEBUF
- FREEPOOL
- GETBUF
- GETPOOL

The following are not supported for BDAM and may cause unpredictable results:

- Callers in 31-bit addressing mode using record format of variable spanned.
- Callers in 31-bit addressing mode using dynamic buffering.
- Callers in 31-bit addressing mode using BSAM to create a BDAM data set.

Addressing modes

Your program can issue all non-VSAM macros in 24-bit addressing mode in task mode. You cannot issue them in SRB mode. Many non-VSAM macros can also be issued in 31-bit addressing mode. When you issue a macro in 24-bit mode, data referred to by the macro must reside below the 16 MB line. When you issue a macro in 31-bit mode, all addresses in registers and 4-byte fields must contain valid 31-bit values although they might point below the 16 MB line. The macro description will state whether it can be issued in 31-bit addressing mode and whether any input fields might reside above the 16 MB line.

If your program supports execution in both 31-bit and 64-bit, then you should precede each of the 64-bit portions of your program with a call to the SYSSTATE macro with AMODE64=YES. The 31-bit portions should be preceded by SYSSTATE with AMODE64=NO. That causes macros to expand correctly.

None of the executable BAM macros support being called in cross memory mode. This means that the invocation must be in primary mode and not AR mode and the primary and secondary address spaces must be the same as the home address space.

Central storage addresses

Storage for all data areas and control blocks can be backed above the 2 GB bar even if your program is running in 24-bit mode. This means that you can code either of the following:

- LOC=(BELOW,ANY)
- LOC=(BELOW,64)

on the GETMAIN or STORAGE macro.

Recommendation: Code the second value as 64.

Note: Code LOC=(BELOW,ANY) or LOC=(BELOW,24) when using tape devices that do not support 64-bit IDAWs.

How to supply an exit routine above 16 MB

Figure 4 on page 153 is an example of a technique to have a 31 bit exit routine residing above the 16 MB line but with an entry point below the line. It is also an example of a glue routine.

The DCB, DCB exit list and entry point for an exit list must reside below the 16 MB line. The OPEN parameter list in this example must reside below the line.

```
BigProg  AMODE 31        Execute in 31-bit addressing mode
BigProg  RMODE ANY       Reside above the 16 MB line
         STORAGE OBTAIN,LENGTH=LenArea,LOC=(BELOW,64)  Get DCB & etc. area
         LR    R2,R1              Load work area base register
         USING WorkArea,R2
         MVC   MyDCB,ModelDCB     Create DCB below the line
         LA    R0,EXL             Point the DCB to exit list below
         STCM  R0,B'0111',DCBEXLSA-IHADCB+MyDCB    the line
         MVI   EXL,X'85'          Set last entry & DCB OPEN exit list
         LA    R0,OPEN24          Point the exit list to the exit rtn
         STCM  R0,B'0111',EXLOPEN   that is below the line
         MVC   OPEN24,ModOPEN24   Move glue code to below line
         LA    R0,OPEN31          Show the 24-bit code where the
         ST    R0,AdOPEN31          31-bit code is above the line
         OI    AdOPEN31,X'80'     Set bit 0 to AMODE 31 in address
         MVC   OpenList,ModelOPEN  Build OPEN parameter list
         OPEN  (MyDCB),MF=(E,OpenList)     List is below the line
          .
          .
         BR    R14                Return to caller
OPEN31   EQU   *         Entry point of DCB OPEN exit above the line
          .
          .
         BSM   0,R14     Switch to 24-bit mode and return to OPEN
ModelOPEN OPEN (,INPUT),MF=L       Model OPEN parameter list
LenOpen  EQU   *-ModelOPEN
* The following is the model for the DCB OPEN exit routine entry point.
* We copy this code to the work area, which is below the line.  The
* BSM sets the current addressing mode (24) in bit 0 of R14 without
* changing anything else in R14.  It also switches to 31-bit due to
* bit 0 in R15 and branches to the address in R15.
ModOPEN24 L    R15,AdOPEN31-OPEN24(,R15) Entry pt to DCB OPEN exit rtn
         BSM   R14,R15        Save AMODE, switch to 31-bit and branch
LenOPEN24 EQU  *-ModOPEN24
* DCB model, which is above the line.
ModelDCB DCB   DSORG=PS,DDNAME=SYSIN,MACRF=(GL,PL)
*
* Dynamic storage that must reside below 16MB line due to DCB & exit
* list restrictions.
WorkArea DSECT
MyDCB    DS    XL(DCBLngQS)  Actual QSAM DCB
* Each entry in DCB exit list is four bytes.
EXL      DC    X         Last entry in exit list and for DCB OPEN exit
EXLOPEN  DS    AL3       Address of 24-bit DCB OPEN exit routine
OPENList DS    XL(LenOpen) OPEN parameter list
* The following is executable code to branch above the 16 MB line.
OPEN24   DS    XL(LenOPEN24)   DCB OPEN exit below 16 MB line
AdOPEN31 DS    A         Address of DCB OPEN exit above the line
LenArea  EQU   *-WorkArea
         DCBD  DSORG=QS,DEVD=DA  Mapping macro for DCB
```

Figure 4. Using a DCB exit list when the application is above the line.

DD statements and dynamic allocation

You can define or allocate data sets by using dynamic allocation, which is SVC 99. Some macro descriptions refer to various keyword parameters that can be coded on a DD JCL statement. All of them have equivalents that can be specified in a call to dynamic allocation (SVC 99) or in a TSO ALLOCATE command. The macros treat all three sources the same.

BSAM, BPAM, and QSAM support the NOCAPTURE, XTIOT, and DSAB above the line options of dynamic allocation. Using these three options of dynamic allocation reduces the overhead of dynamic allocation and reduces virtual storage usage below the 16 MB line. Dynamic allocation is described in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Note: Although the NOCAPTURE dynamic allocation option does not have DD statement or TSO ALLOCATE option equivalents, the main non-VSAM access methods (BPAM, BSAM, and QSAM) do support the NOCAPTURE option of dynamic allocation (SVC 99).

Chapter 5. Non-VSAM macro descriptions

This chapter contains non-VSAM macro formats. See "BAM macro instructions" on page 149 for more information.

BLDL—Build a directory entry list (BPAM)

The BLDL macro is used to obtain a list of information from the directory of a partitioned data set or partitioned data set extended (PDSE). The problem program must supply a storage area that includes information about:

- The number of entries in the list
- The length of each entry
- The name of each member (or alias) to be searched for.
- The caller may optionally supply a list prefix area. An 8 byte list prefix is required if any of the following optional parameters are specified:

  - NOCONNECT

  - BYPASSLLA

  - START=

  - STOP=

Member and alias names in the list must be in alphanumeric order. You must test all read and write operations using the same data control block for completion before issuing the BLDL macro.

The BLDL macro establishes a connection to each PDSE member when it is found in the PDSE directory, unless the NOCONNECT option is used. The connection remains until the PDSE is closed. See *z/OS DFSMS Using Data Sets* for more information on the BLDL macro and PDSE connections.

The BLDL macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The format of the BLDL macro is:

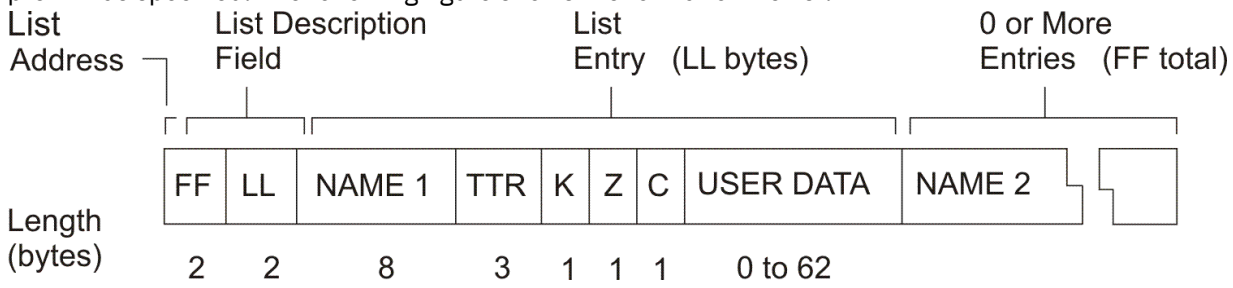
```
[label]          BLDL          dcb address
                               ,list address
                               [,NOCONNECT]
                               [,BYPASSLLA]
                               [,START=concat#]
                               [,STOP=concat#]
```

***dcb address*—RX-Type Address, (2-12) or (1)**
specifies the address of an open data control block (DCB). The DCB must be opened to a partitioned data set, a PDSE, a z/OS UNIX directory or a concatenation of any combination of them. You can specify zero to indicate that the data set search order begins with the task libraries, then proceeds to the job library or step library (whichever is active) followed by the link list libraries.

If you specify a non-zero DCB address and a requested member is not found in the partitioned data set, UNIX directory or concatenation to which the DCB is open, then the search for that member will stop; the job library, step library, task libraries or link list libraries will not be searched.

***list address*—RX-Type Address, (2-12), or (0)**
specifies the address of the list completed when the BLDL macro is issued. The list must be on a halfword boundary. When BLDL is issued in 31-bit addressing mode, the list may reside above the

16MB line. The list address points to the FF field of the parameter list without regard to whether a prefix was specified. The following figure shows the format of the list:



FF: This field must contain a binary value indicating the total number of entries in the list.

LL: This field must contain a binary value indicating the length, in bytes, of each entry in the list. If the exact length of the entry is known, specify the exact length. Otherwise, specify at least 62 bytes (decimal) if an entry in the list is to be used with an ATTACH, ATTACHX, LINK, LINKX, LOAD, XTL, or XCTLX macro. To ensure that your program reads the entire directory entry, provide 76 bytes for each entry. The minimum length for a list is 12 bytes.

NAME: This field must contain the member name or alias to be located. The name must start in the first byte of the name field and be padded to the right with blanks (if necessary) to fill the 8-byte field.

When the BLDL macro is executed, 5 fields of the directory entry list are filled in by the system. The specified length (LL) must be at least 14 bytes to fill in the Z and C fields. If the LL field is 12 bytes, only the NAME, TT, R, and K fields are returned. The 5 fields are:

TT: Indicates the two-byte relative track number where the beginning of the member is located.

R: Indicates the one-byte relative block (record) number on the track indicated by TT.

For a PDSE or a UNIX directory, TTR is a token that does not represent the physical location of the member in the data set.

K: Indicates the concatenation number of the data set. For the first or only data set, this value is zero.

Z: Indicates where the system found the directory entry:

Code
Meaning

0
Private library

1
Link library

2
Job, task, or step library

3-16
Job, task, or step library of parent task n, where n = Z-2

C: Indicates the type of name (primary or alias) for the number of note list fields (TTRNs), and the length of the user data field (indicated in halfwords). The following describes the meaning of the 8 bits:

Bit
Meaning

0=0
Indicates a member name.

0=1
Indicates an alias.

1-2
Indicates the number of TTRN fields (maximum of 3) in the user data field.

3-7
Indicates the total number of half words in the user data field.

USER DATA: The user data field contains the user data from the directory entry. If the length of the user data field in the BLDL list is equal to or greater than the user data field of the directory entry, the entire user data field is entered in the list. Otherwise, the list contains only the user data for which there is space.

NOCONNECT
specifies that the PDSE member is not to be connected. When issuing BLDL, you must provide a prefix of 8 bytes that immediately precedes the list of member names. The BLDL macro expansion will clear and initialize the prefix. The listaddr parameter must point to the FF field.

BYPASSLLA
specifies that the search for members should not include searching Library Lookaside.

START=concat byte address —RS-type address, (2-12)
specifies either the address of a byte which contains the concatenation number for the first data set to be searched in the concatenation or it specifies a register containing the number (not the address of the number). The concatenation number is relative to zero.

STOP=concat byte address —RS-type address, (2-12)
specifies either the address of a byte which contains the concatenation number for the last data set to be searched in the concatenation or it specifies a register containing the number (not the address of the number). The concatenation number is relative to zero.

Completion codes

When the system returns control to the problem program, the low-order byte of register 15 contains a return code. The low-order byte of register 0 contains a reason code.

The BLDL return and reason codes are:

Table 28. BLDL Completion Codes

Return Code (15)	Reason Code (0)	Meaning
00 (X'00')	00 (X'00')	Successful completion.
04 (X'04')	00 (X'00')	One or more entries in the list could not be filled; the list supplied can be invalid (the list length was less than 12 or the number of entries was zero or negative). If a search is attempted but the entry is not found, the R field (byte 11) for that entry is set to zero.
04 (X'04')	04 (X'04')	An attempt to connect to a UNIX file failed because at least one data set in the concatenation is protected with RACF execute-only authority.
08 (X'08')	00 (X'00')	A permanent I/O error was detected when the system attempted to search the directory.
08 (X'08')	04 (X'04')	Insufficient virtual storage was available.
08 (X'08')	08 (X'08')	Invalid data extent block (DEB), or the DEB is not owned by a TCB in the current family of TCBs, or the UCB address in the DEB is zero (this indicates a dummy data set.
08 (X'08')	20 (X'14')	An error was returned by IGGSOOPN when attempting to connect to a UNIX file. See message IEC104I for more details.
08 (X'08')	24 (X'18')	An attempt to connect to a UNIX file failed because the user did not have RACF authority to access to the file.

Table 28. BLDL Completion Codes (continued)

Return Code (15)	Reason Code (0)	Meaning
08 (X'08')	20 (X'14')	START= value is greater than either the specified STOP= value or the highest concatenation number for the DCB.
08 (X'08')	32 (X'24')	START= or STOP= was specified with a DCB address of zero.
08 (X'08')	36 (X'28')	STOP= value is greater than the highest concatenation number for the DCB.

BSP—Backspace a physical record (BPAM, BSAM—magnetic tape and DASD only)

The BSP macro backspaces the current volume one data block (physical record). All input and output operations must be tested for completion before the BSP macro is issued. You can use the BSP macro only with a BSAM or BPAM DCB. You can use the BSP macro on a data set created by QSAM if it is opened using BSAM. Do not use the BSP macro if the CNTRL, NOTE, or POINT macro is being used (see the discussion of UNIX files, below, for NOTE and POINT exceptions).

Any attempt to backspace across a file mark results in a return code of X'04' and your tape or direct access volume is not positioned after the file mark. This means you cannot issue a successful BSP macro after your EODAD routine is entered unless you first reposition the tape or direct access volume into your data set. (Use CLOSE TYPE=T to position to the end of your data set.)

PDSE You can use the BSP macro to backspace the current member one simulated block. You can then reread or rewrite the simulated block. However, you cannot backspace beyond the start of a PDSE directory nor backspace beyond the start of a PDSE member. See the chapter on PDSEs in *z/OS DFSMS Using Data Sets* for information on using the BSP macro with variable spanned and variable blocked spanned records.

Extended format data sets: The system treats the stripes of a striped data set as one volume. If it is a compressed format data set, the amount of data backspaced over is what was originally written by one WRITE macro or simulated by PUT macros as a block.

UNIX files: BSP is supported for UNIX files (except for FIFO or character special files or with PATHOPTS=OAPPEND) by positioning you to the beginning of the block which was just read or written.

- BSP can only be issued following the completion of a successful CHECK (for READ or WRITE), NOTE, or CLOSE TYPE=T LEAVE request. (A BSP cannot be followed by another BSP). A BSP following a request other than those listed above gives a return code of X'04' and a reason code of X'0E'.
- A BSP issued for a FIFO or character special file gives a return code of X'04' and a reason code of X'0F'.
- A BSP issued for a UNIX file opened with PATHOPTS=OAPPEND gives a return code of X'04' and a reason code of X'10'.

Magnetic Tape: A backspace is always made toward the beginning of the tape.

SYSIN or SYSOUT Data Sets: A BSP macro is ignored, but a completion code is returned.

The BSP macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The format of the BSP macro is:

[*label*]	BSP	*dcb address*

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for the volume to be backspaced. You must open the data set on the volume to be backspaced before issuing the BSP macro. When issued in 31-bit addressing mode, the input DCB address must be a clean 31-bit address.

Completion codes

When the system returns control to the problem program, the low-order byte of register 15 contains a return code. The low-order byte of register 0 contains a reason code.

The BSP return and reason codes are:

Return Code (15)	Reason Code (0)	Meaning
00 (X'00')	00 (X'00')	Successful completion.
04 (X'04')	01 (X'01')	A backspacing request was ignored on a SYSIN or SYSOUT data set.
04 (X'04')	02 (X'02')	Backspace not supported for this device type.
04 (X'04')	03 (X'03')	Backspace failed; insufficient virtual storage was available.
04 (X'04')	04 (X'04')	Backspace failed; permanent I/O error.
04 (X'04')	05 (X'05')	Backspace into load point or beyond start of data set on the current volume.
04 (X'04')	06 (X'06')	The supplied DCB or its DEB is invalid.
04 (X'04')	07 (X'07')	Backspace detected an invalid extent value (M).
04 (X'04')	08 (X'08')	Backspace issued while I/O was in progress.
04 (X'04')	09 (X'09')	Backspace was attempted within a PDSE directory.
04 (X'04')	10 (X'0A')	Backspace failed; backspace past the start of a PDSE member is not allowed.
04 (X'04')	11 (X'0B')	Backspace failed; system control block used for PDSE processing contains incorrect information. This is a likely system logic error.
04 (X'04')	12 (X'0C')	SMS error occurred while processing a PDSE member with variable blocked records.
04 (X'04')	13 (X'0D')	Backspace failed; system control block used for processing extended format data sets contains incorrect information.
04 (X'04')	14 (X'0E')	Backspace failed for a UNIX file. Backspace was issued following a macro request other than CHECK, NOTE, or CLOSE TYPE=T LEAVE.
04 (X'04')	15 (X'0F')	Backspace failed. Backspace issued for a FIFO or character special file is not allowed.
04 (X'04')	16 (X'10')	Backspace failed. Backspace issued for a UNIX file opened with PATHOPTS=OAPPEND is not allowed.
08 (X'08')	01 (X'01')	Backspace not successful; internal system error occurred while processing a PDSE.

BUILD—Build a buffer pool (BDAM, BISAM, BPAM, BSAM, QISAM, and QSAM)

The BUILD macro is used to construct a buffer pool in an area provided by the problem program. The buffer pool can be used by more than one data set through separate data control blocks. For BDAM, BISAM, BPAM and BSAM your program can obtain individual buffers from the buffer pool using the GETBUF macro, and return them to the buffer pool using a FREEBUF macro. For QISAM and QSAM, OPEN obtains buffers from and CLOSE returns buffers to the buffer pool. See *z/OS DFSMS Using Data Sets* for an explanation of the interaction of the DCB, BUILD, and GETBUF macros in each access method, and the buffer size requirements.

The BUILD macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

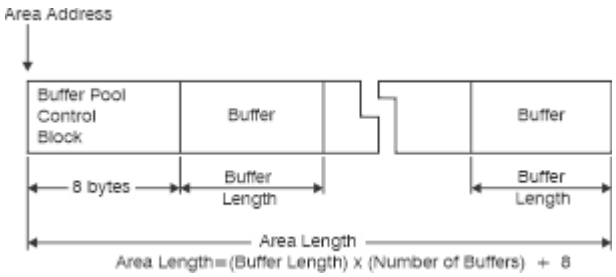
Note: BSAM cannot support 64 bit real storage for any tape devices that do not support 64 bit IDAWs. Applications that use BSAM to process a tape data set can experience ABEND0D3 and/or ABENDB00 due to the inability to use a 64-bit IDAW by the device.

The format of the BUILD macro is:

[*label*]	BUILD	*area address* ,{*number of buffers*,*buffer length* | (0)}

***area address*—RX-Type Address, (2-12), or (1)**
specifies the address of the area to be used as a buffer pool. The area must start on a fullword boundary. When issued in 31-bit addressing mode, the input area address must be a clean 31-bit address. If the area resides above the line, it cannot be used by other access method macros.

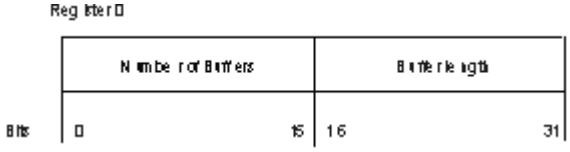
The following illustration shows the format of the buffer pool:



***number of buffers*—symbol, decimal digit, absexp, or (2-12)**
specifies the number of buffers in the buffer pool to a maximum of 255.

***buffer length*—symbol, decimal digit, absexp, or (2-12)**
specifies the length, in bytes, of each buffer in the buffer pool. If the value specified for the buffer length is not a multiple of four the system rounds the value specified to the next higher multiple of four. The maximum length that can be specified is 32 760 bytes. For QSAM, the buffer length must be at least as large as the value specified in the block size (DCBBLKSI) field of the data control block.

(0)
The number of buffers and buffer length can be specified in general register 0. The following illustration shows that if (0) is coded, register 0 must contain the binary values for the number of buffers and buffer length.



BUILDRCD—Build a buffer pool and a record area (QSAM)

The BUILDRCD macro builds a buffer pool and a record area in an area of storage you provide. This macro is used only for variable-length, spanned records processed in QSAM locate mode. If the extended logical record interface (XLRI) is used to process RECFM=DS or RECFM=DBS records (ISO/ANSI/FIPS variable spanned or variable blocked spanned), you can use the BUILDRCD macro to build a record area to a maximum length of 16777183 bytes. Using this macro before the data set is opened, or before the end of the DCB open exit routine, provides a buffer pool that can be used for a logical record interface rather than a segment interface for variable-length spanned records. To invoke a logical record interface, specify BFTEK=A in the data control block (DCB). You cannot specify the BUILDRCD macro when logical records exceed 32 760 bytes.

You must release the buffer pool and the record area after issuing a CLOSE macro for all the data control blocks that use the buffer pool and the record area.

The BUILDRCD macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The standard form of the BUILDRCD macro is as follows (the list and execute forms are shown following the description of the standard form):

[*label*]	BUILDRCD	*area address* ,*number of buffers* ,*buffer length* ,*record area address* [,*record area length*]

***area address*—A-Type Address or (2-12)**
specifies the address of the area used as a buffer pool. The area must start on a fullword boundary. When issued in 31-bit addressing mode, the input area address must be a clean 31-bit address and it must reside below the line. BUILDRCD does not support buffers above the line.

area length = [(buffer length) * (number of buffers) + 12]

***number of buffers*—symbol, decimal digit, absexp, or (2-12)**
specifies the number of buffers, to a maximum of 255, in the buffer pool.

***buffer length*—symbol, decimal digit, absexp, or (2-12)**
specifies the length, in bytes, of each buffer in the buffer pool. The value specified for the buffer length must be a fullword multiple; otherwise, the system rounds the value specified to the next higher fullword multiple. The maximum length that can be specified is 32 760 bytes.

***record area address*—A-Type Address or (2-12)**
specifies the address of the storage area that is used as a record area. The area must start on a doubleword boundary and have a length of the maximum logical record (LRECL) plus 32 bytes. When issued in 31-bit addressing mode, the record area address must be a clean 31-bit address and it must reside below the line. BUILDRCD does not support buffers above the line.

***record area length*—symbol, decimal digit, absexp, or (2-12)**
specifies the length of the record area that is used. The area must be as long as the maximum length logical record plus 32 bytes for control information. If the *record area length* is omitted, the problem program must store the *record area length* in the first 4 bytes of the record area.

BUILDRCD—List form

The list form of the BUILDRCD macro is used to construct a program parameter list. The description of the standard form of the BUILDRCD macro explains the function of each parameter. The format description below indicates the optional and required parameters in the list form only.

The list form of the BUILDRCD macro is:

[*label*]	BUILDRCD	*area address* ,*number of buffers* ,*buffer length* ,*record area address* [,*record area length*] ,MF=L

area address—A-Type Address

number of buffers—symbol, decimal digit, or absexp

buffer length—symbol, decimal digit, or absexp

record area address—A-Type Address

record area length—symbol, decimal digit, or absexp

MF=L
specifies that the BUILDRCD macro is used to create a parameter list that is referred to by an execute form instruction.

You can construct a parameter list by coding only MF=L (without the preceding comma). In this case, the list is constructed for the area address, number of buffers, buffer length, and record area address parameters. If the *record area length* is also required, code the parameters as follows:

```
[label] BUILDRCD,,,,0,MF=L
```

The preceding example shows the coding to construct a list containing address constants with a value of 0 in each constant. The actual values can then be supplied by the execute form of the BUILDRCD macro.

BUILDRCD—Execute form

A remote parameter list is referred to, and can be modified by, the execute form of the BUILDRCD macro. The description of the standard form of the BUILDRCD macro explains the function of each parameter. The format description below indicates the optional and required parameters for the execute form only.

The execute form of the BUILDRCD macro is:

[*label*]	BUILDRCD	[*area address*] ,[*number of buffers*] ,[*buffer length*] ,[*record area address*] [,*record area length*] ,MF=(E,*list address*)

area address—RX-Type Address or (2-12)

number of buffers—absexp

buffer length—absexp

record area address—RX-Type Address or (2-12)

record area length—absexp

MF=(E,*list address*)
specifies that the execute form of the BUILDRCD macro is used, and an existing parameter list (created by a list-form instruction) is used. MF is coded as follows:

E

list address—RX-Type Address, (2-12), or (1)

CHECK—Wait for completion of a request (BDAM, BISAM, BPAM, and BSAM)

The CHECK macro places the active task in the wait condition, if necessary, until the associated input or output operation is completed. The input or output operation is then tested for errors and exceptional conditions. If the operation completes successfully, control is returned to the instruction following the CHECK macro. If the operation does not complete successfully, the error analysis (SYNAD) routine or end-of-data (EODAD) routine is given control. If the appropriate routine is not provided, the task is abnormally terminated. These routines are discussed in the SYNAD and EODAD parameters of the DCB and DCBE macros.

The following conditions are also handled for BPAM and BSAM only:

When Reading: The end-of-data (EODAD) routine is given control if an input request is made after all the records are retrieved. Volume switching is automatic for a multivolume data set not opened for UPDAT. For a multivolume data set opened for UPDAT, the end-of-data routine is entered at the end of each volume. The system treats a striped data set as a single volume.

When Writing: Additional space on the device is obtained when the current space is filled and more WRITE macros have been issued.

When writing on a cartridge tape, CHECK ensures that the data has been transferred to the tape subsystem and not necessarily to tape. To ensure that all of the data is on the tape, issue either a CLOSE macro or a SYNCDEV macro with INQ=NO. However, SYNCDEV generally is not useful and gives poor performance. Without SYNCDEV, if any data fails to get on the tape, a subsequent CHECK macro or CLOSE macro will detect and handle the I/O error.

You must issue a CHECK, WAIT, or EVENTS macro for each input and output operation. For BSAM and BPAM, the CHECK, WAIT, or EVENTS macros must be issued in the same order as the READ or WRITE macros were issued for the data set. For information on when you can use the WAIT or EVENTS macro, see *z/OS DFSMS Using Data Sets*.

Processing PDSEs: If a PDSE member is open for update and in a storage class with "Guaranteed Synchronous Write" specified, a CHECK macro issued following a WRITE macro guarantees that the data is synchronized to DASD. Otherwise, synchronization is not guaranteed until CLOSE, or the STOW macro or the SYNCDEV macro is issued. Specifying "Guaranteed Synchronous Write" in the storage class produces the same result as issuing the SYNCDEV macro after every CHECK. On output, CHECK guarantees that the ECB is posted and that the data has been moved from your buffer into an internal system buffer, allowing your buffer to be available for reuse.

Processing UNIX files: CHECK guarantees that the ECB is posted and that any output data has been moved from your buffer to an internal system buffer, allowing your buffer to be available for reuse.

CHECK does not necessarily guarantee that the output data has been synchronized to the output file, unless PATHOPTS=OSYNC is specified. If PATHOPTS=OSYNC is specified, CHECK guarantees that the output data has been synchronized to the output file. Issuing the CLOSE or the SYNCDEV macro guarantees that all output data has been synchronized to the output file.

Processing Compressed Format Data Sets: When processing a compressed format data set on output, CHECK guarantees that the ECB is posted and that the data has been moved from your buffer into an internal system buffer, allowing your buffer to be available for reuse. CHECK does not guarantee that the data is synchronized to DASD. Synchronization is not guaranteed until CLOSE or the SYNCDEV macro is issued. Specifying "Guaranteed Synchronous Write" in the storage class produces the same result as issuing the SYNCDEV macro after every CHECK.

Data Conversion: You can request conversion by coding LABEL=(,AL) or (,AUL) in the DD statement, or by coding OPTCD=Q in the DCB macro or DCB subparameter of the DD statement. If conversion is requested, the check routine automatically converts BSAM records, as they are read, from one character representation to another if the record format is F, FB, D, DB, or U. Conversion occurs when the check

routine determines that the input buffer is full. Conversion is performed according to one of the following techniques:

- **Coded Character Set Identifier (CCSID) Conversion.** If CCSIDs are supplied from any source[1] for ISO/ANSI V4 tapes, records are converted between the CCSID which represents the data on tape and the CCSID as seen by the problem program. You can also prevent conversion by supplying a special CCSID.
- **Default Character Conversion.** If you are using non-ISO/ANSI V4 tapes or if CCSIDs are not supplied by any source, data management converts the records between ASCII code and EBCDIC code using specific tables defined for this default character conversion.

Refer to *z/OS DFSMS Using Data Sets* for a complete description of CCSID conversion and default character conversion.

After you issue a CHECK macro when reading format D or DB blocks without BUFUFFEL, the length of the block is in the DCB LRECL field in the DCB. It will remain valid until the next CHECK macro.

The CHECK macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The format of the CHECK macro is:

[*label*]	CHECK	*decb address* [,DSORG={IS|ALL}]

***decb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data event control block created or used by the associated READ or WRITE macro. When issued in 31-bit addressing mode, the input DECB address must be a clean 31-bit address. If your SYNAD or EODAD routine is entered, it is entered in the addressing mode in which the CHECK was issued. If you supplied a SYNAD or EODAD routine which resides above the line in the DCBE, then the CHECK must be issued in 31-bit addressing mode.

DSORG={IS|ALL}
specifies the type of data set organization. You can specify:

IS
specifies that the macro expansion is for BISAM use only.

ALL
specifies that the macro expansion is for BDAM, BISAM, BPAM, or BSAM use.

If DSORG is omitted, the macro expansion is for BDAM, BPAM, or BSAM use only.

CHKPT—Take a checkpoint for restart within a job step

The CHKPT macro is coded inline in the problem program. When this macro executes, the operating system writes a checkpoint entry in a checkpoint data set. The entry consists of job step information, such as virtual-storage data areas, data set position, and supervisor control, from the problem program.

After the checkpoint information has been written, control is returned to the instruction following the CHKPT macro.

When an application program takes a checkpoint, the system records information about the status of that program in a checkpoint data set. This information includes the location on disk or tape where the application is currently reading or writing each open data set. If a data set that is open at the time of the checkpoint is moved to another location before the restart, you cannot restart the application from the checkpoint because the location-dependent information recorded by checkpoint/restart is no longer valid.

[1] CCSID may be supplied in the CCSID subparameter of a JOB, EXEC, or DD statement or the tape label.

There are several system functions (for example, DFSMShsm or DFSMSdss) that might automatically move a data set without the owner specifically requesting it. To ensure that all checkpointed data sets remain available for restart, the checkpoint function sets the unmovable attribute for each SMS-managed sequential data set that is open during the checkpoint. An exception is the data set containing the actual recorded checkpoint information (the checkpoint data set), which does not require the unmovable attribute.

You can move checkpointed data sets when you no longer need them to perform a restart. The DFSMShsm and DFSMSdss FORCECP(days) commend allow you to use operations such as migrate, copy, or defrag to move an SMS-managed sequential data set based on a number of days since the last access. DFSMShsm recall and DFSMSdss restore and copy are operations that turn off the unmovable attribute for the target data set.

For information about the CHKPT macro, see *z/OS DFSMSdfp Checkpoint/Restart*.

CLOSE—Disconnect program and data (BDAM, BISAM, BPAM, BSAM, QISAM, and QSAM)

The CLOSE macro creates output data set labels and allows you to position volumes. The fields of the data control block (DCB) and DCBE are restored to the condition that existed before the OPEN macro was issued, and the data set is disconnected from the processing program. You can specify final volume positioning or disposition for the current volume to override the positioning implied by the DISP parameter of the DD statement. Any number of *dcb address* parameters and associated options can be specified in the CLOSE macro.

After a CLOSE has been issued for several data sets, a return code of 4 indicates that at least one of the data sets, VSAM or non-VSAM, was not closed successfully.

A FREEPOOL macro should normally follow a CLOSE macro (without TYPE=T) to regain the buffer pool storage space if OPEN or GETPOOL built the buffer pool. This also allows a new buffer pool to be built if the DCB is reopened with different record size attributes. However, if you requested via the DCBE that OPEN obtain QSAM buffers above the line, CLOSE frees the buffer pool obtained by OPEN. Therefore, in this case, a FREEPOOL macro is not required following the CLOSE macro.

Associated data sets for an IBM 3525 Card Punch can be closed in any sequence, but, if one data set is closed, I/O operations cannot be initiated for any of its associated data sets. Additional information about closing associated data sets is contained in *z/OS DFSMS Using Data Sets*.

A special parameter, TYPE=T, temporary close, is provided for processing with BSAM.

The CLOSE macro does not support more than a total of 255 spooled, SUBSYS or compressed format data sets, for one invocation.

If you use a "reserved" relative generation number character as the first character of a member name, the stow will not occur, you must issue your own stow.

Extended format data sets: If you request release of unused space for extended format data sets, CLOSE releases space on each stripe if possible. After the space is released, the size of some stripes may differ slightly from others. Depending on the unit used for allocation, the difference will be at most one track or cylinder.

When a compressed format data set is written using BSAM or QSAM, the CLOSE macro ensures that all data has been synchronized to DASD.

PDS and PDSE data sets: If the PDS or PDSE is open for OUTPUT, OUTIN or INOUT, and a member name was specified in the JCL (or JFCB with OPEN TYPE=J), and the last operation was either a WRITE (BSAM) or a PUT (QSAM), and the application has not issued its own STOW, then CLOSE will issue a STOW. However, if the first character of the member name is '+' (X'4E'), '-' (X'60'), or (X'Fx'), CLOSE will not issue STOW. This is because these characters identify the generation data set (GDS) within a generation data group (GDG), and the access method uses the first character of the member name field in the JFCB to distinguish between a GDS identifier and a member name.

PDSEs: After PDSE members are written or updated using BSAM or QSAM, the CLOSE macro synchronizes member data to DASD.

UNIX files: When a file is written using BSAM or QSAM, the CLOSE macro ensures that all data has been synchronized to the file.

SMF records: CLOSE does not write SMF type 14/15 records for UNIX files. DFSMS relies on z/OS UNIX to write appropriate SMF records when requested by the system programmer.

The CLOSE macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The standard form of the CLOSE macro is as follows (the list and execute forms are shown following the description of the standard form):

[*label*]	CLOSE	(*dcb address*[,[*option*][,...]]) [,TYPE=T] [,MODE=24|31]

***dcb address*—A-Type Address or (2-12)**
specifies the address of the data control block for the opened data set to be closed.

Requirement: If the register format is used, then the register must be enclosed within parentheses. For example, CLOSE ((2)).

option
Each of these options indicates the volume positioning to occur when the data set is closed. These options are generally used with TYPE=T for data sets on magnetic tape. However, options specified in the CLOSE macro override disposition specifications in the JCL for all data sets. The options are:

REREAD
specifies that the current volume is to be positioned to reprocess the data set. If processing was forward, the volume is positioned to the beginning of the data set. If processing was backward (RDBACK), the volume is positioned to the end of the data set. If FREE=CLOSE is specified in the JCL, the data set is not unallocated until the end of the job step.

LEAVE
specifies that the current volume is to be positioned to the logical end of the data set. If processing was forward, the volume is positioned to the end of the portion of the data set residing on the current volume. If processing was backward (RDBACK), the volume is positioned to the beginning of the portion of the data set residing on the current volume. If FREE=CLOSE is specified in the JCL, the data set is not unallocated until the end of the job step.

REWIND
specifies that the current magnetic tape volume is to be positioned at the load point, regardless of the direction of processing. REWIND cannot be specified when TYPE=T is specified. If FREE=CLOSE is coded on the DD statement associated with the data set being closed, coding the REWIND option frees the data set when it is closed rather than at the end of the job step.

FREE
specifies that the current data set is freed when the data set is closed, rather than when the job step terminates. For tape data sets, this means that the volume is eligible for use by other tasks or to be demounted. Direct access volumes can also be freed for use by other tasks. They can be freed for demounting if (1) no other data sets on the volume are open and (2) the volume is otherwise demountable. Do not use this option with CLOSE TYPE=T. (For other restrictions on the FREE parameter, see *z/OS MVS JCL Reference*.)

DISP
specifies that a tape volume is to be disposed of in the manner implied by the DD statement associated with the data set. Direct access volume positioning and disposition are not affected by

this parameter. There are several dispositions that can be specified in the DISP parameter of the DD statement; DISP can be PASS, DELETE, KEEP, CATLG, or UNCATLG.

Depending on how the DISP option is coded in the DD statement, the current magnetic tape volume is positioned as follows:

DISP Parameter	Action
PASS	Forward space to the end of data set on the current volume.
DELETE	Rewind the current volume.
KEEP, CATLG, or UNCATLG	The volume is rewound and unloaded, if necessary.

If FREE=CLOSE is coded in the DD statement associated with this data set, coding the DISP option in the CLOSE macro results in the data set being freed when the data set is closed, rather than at the time the job step is terminated.

When the option subparameter is omitted, DISP is assumed. For TYPE=T, this is processed as LEAVE during execution. The LEAVE and REREAD options are used only for magnetic tape or CLOSE TYPE=T.

TYPE=T

You can code CLOSE TYPE=T to temporarily close sequential data sets on magnetic tape and direct access volumes processed with BSAM. When you use TYPE=T, the DCB used to process the data set maintains its open status, and you should not issue another OPEN macro to continue processing the same data set. This option cannot be used in a SYNAD exit routine.

TYPE=T causes the system control program to process labels, modify some of the fields in the system control blocks for that data set, and reposition the volume (or current volume for multivolume data sets) in much the same way that the normal CLOSE macro does.

When you code TYPE=T, you can specify that the volume either be positioned at the end of data (the LEAVE option) or be repositioned at the beginning of data (the REREAD option). Magnetic tape or DASD volumes are repositioned either immediately before the first data record or immediately after the last data record. The presence of tape labels has no effect on repositioning.

For PDSEs and partitioned data sets, CLOSE TYPE=T does no operation except when reading the PDSE or partitioned data set directory sequentially. If you code CLOSE TYPE=T with the REREAD option, the data set is repositioned to the beginning of the directory.

If you code the RLSE keyword with the SPACE parameter on the DD statement that describes the output data set, it is ignored by temporary close (CLOSE TYPE=T). Unused space is released only if the data set is OPEN with the OUTPUT, OUTIN, INOUT, EXTEND or OUTINX option, and the last operation was OPEN, WRITE (and CHECK), PUT, or CLOSE with TYPE=T.

For extended format data sets open for output, CLOSE TYPE=T updates the data set label for each stripe to correctly reflect the used space on each volume.

While an extended format data set is open for output, the data set labels do not correctly reflect the used space in the data set. An open to an input or update DCB (while the output DCB is still open for output) will not reflect the correct data set size of an extended format data set. You may choose to issue a CLOSE LEAVE,TYPE=T on the output DCB to cause subsequent OPENs for input/update to reflect the correct amount of used space. CLOSE TYPE=T is ignored for a FIFO and character special UNIX file.

MODE=<u>24</u>|31

You can code CLOSE MODE=31 to specify a long form parameter list that can contain 31-bit addresses. The default, MODE=24, specifies a short form parameter list with 24-bit addresses. Your program does not need to be executing in 31-bit addressing mode to use MODE=31 in the CLOSE macro. This parameter specifies the form of the parameter list, not the addressing mode of the program.

The caller of the standard form of the macro with the short form of the parameter list must reside below the 16MB line, but the caller can be executing in 31-bit mode. All access method control blocks (ACBs) and DCBs are below the 16MB line.

The long form parameter list can reside above or below the 16MB line. Although the access method control block (ACB) or DCB address is contained in a 4-byte field, the DCB must be below the 16MB line. Except for VSAM or Virtual Telecommunications Access Method (VTAM) ACBs, all ACBs must also be below the 16MB line. Therefore, the leading byte of the ACB or DCB address must contain zeros. If the byte contains something other than zeros, an IEC290I message is issued and the data set is not closed.

For additional information and coding restrictions, see *z/OS DFSMS Using Data Sets*.

CLOSE—List form

The list form of the CLOSE macro is used to construct a data management parameter list. Any number of parameters (data control block addresses and associated options) can be specified. A parameter list constructed by a CLOSE macro, list form, can be referred to by either an OPEN or CLOSE execute-form instruction. You must ensure that the MODE parameters on the list and execute forms are consistent. Errors and unpredictable results occur if the modes are inconsistent.

There are two forms of the list, the short form and the long form. The short form list consists of a one-word entry for each DCB or ACB in the parameter list. The high-order byte is used for the options and the 3 low-order bytes are used for the DCB address. The long form list consists of an eight byte entry for each DCB or ACB in the parameter list. The high order byte is used for the options and the low order four bytes are used for the DCB or ACB address. For either form of list, the end of the list is indicated by a 1 in the high-order bit of the last entry's option byte. The length of a list generated by a list-form instruction must be equal to the maximum length required by an execute-form instruction that refers to the same list. You can construct a maximum length list by one of two methods:

- Code a list-form instruction with the maximum number of parameters required by an execute-form instruction that refers to the list.
- Code a maximum length list by using commas in a list-form instruction to acquire a list of the appropriate size. For example, coding CLOSE `(,,,,,,,,,),MF=L` would provide a list of 5 fullwords (5 *dcb addresses* and 5 options).

Entries at the end of the list that are not referred to by the execute-form instruction are assumed to have been filled in when the list was constructed or by a previous execute-form instruction. Before using the execute-form instruction, you can shorten the list by placing a 1 in the high-order bit of the last DCB entry to be processed.

A zeroed work area on a word boundary is equivalent to CLOSE `(,DISP,...),MF=L` and can be used in place of a list-form instruction. Allocate four bytes per entry if you wish the effect of MODE=24. Allocate eight bytes per entry if you wish the effect of MODE=31. The high-order bit of the last DCB entry must contain a 1 before this list can be used with the execute-form instruction.

The list form of the CLOSE macro is:

[*label*]	CLOSE	([*dcb address,*],[*option*],...) [,TYPE=T] [,MF=L [,MODE=24|31]

dcb address—A-Type Address
option—Same as standard form

TYPE=T
can be coded in the list-form instruction to allow the specified option to be checked for validity when the program is assembled.

MF=L
specifies the CLOSE macro is used to create a data management parameter list referred to by an execute-form instruction.

MODE=24|31
coded the same as the standard form. This specification must match that of the execute form.

CLOSE—Execute form

A list form of the CLOSE macro is used in and can be modified by the execute form of the CLOSE macro. The parameter list can be generated by the list form of either an OPEN macro or a CLOSE macro.

The description of the standard form of the CLOSE macro explains the function of each parameter.

The execute form of the CLOSE macro is:

[*label*]	CLOSE	[([*dcb address,*],[*option*],...)] [,TYPE=T] [,MF=(E,*address of list form*) [,MODE=24|31]

dcb address—RX-Type Address or (2-12)

option—If specified, same as the standard form. If not specified, the option specified in the list form of the CLOSE macro is used.

TYPE=T—Same as standard form.

MF=(E,*address of the list form*)
specifies that the execute form of the CLOSE macro is being used, and the parameter list is created by the list form of the CLOSE macro. MF= is coded as described in the following:

E
address of the list form of the CLOSE (or OPEN) macro —RX-Type Address, (2-12), or (1)

MODE=24|31
coded the same as the standard form. This specification must match that of the list form.

CLOSE return codes

When your program receives control after it has issued a CLOSE macro, a return code in register 15 indicates whether all data sets were closed successfully.

The CLOSE return codes are:

Return Code (15)	Meaning
0(X'0')	All data sets were closed successfully.
4(X'4')	At least one data set (VSAM or non-VSAM) was not closed successfully.

Example 1: CLOSE macro

In this example DCB1 is closed.

```
         CLOSE (DCB1)
```

Example 2: CLOSE macro

In this example the DCB that register DCBPTR points to is closed.

```
         CLOSE ((DCBPTR),REWIND)
```

Example 3: CLOSE macro

In this example a 31-bit parameter list with room for two DCBs or ACBs is generated.

```
CLIST    CLOSE (,,,),MF=L,MODE=31
```

CNTRL—Control directly allocated input/output device (BSAM and QSAM)

The CNTRL macro controls magnetic tape drives (BSAM only for a data set that is not open for output), directly allocated card readers, IBM 3525 Card Punches (read and print features), printers (BSAM and QSAM), and the IBM 3890 Document Processor (QSAM only).

The MACRF parameter of the DCB macro must specify a C. The CNTRL macro is ignored for spooled SYSIN or SYSOUT data sets. For BSAM, all input and output operations must be tested for completion before the CNTRL macro is issued. The control facilities available are as follows:

Card Reader: Provides stacker selection, as follows:

QSAM:For unblocked records, issue a CNTRL macro after every input request. For blocked records, issue a CNTRL macro after the last logical record on each card retrieved. In either case, do not issue a CNTRL macro after a GET macro causes control to pass to the EODAD routine. The move mode of the GET macro must be used, and the number of buffers (BUFNO field of the DCB) must be 1. If a CLOSE macro is issued before the last card is read, the operator should clear the reader before the device is used again.

BSAM:The CNTRL macro should be issued after every input request.

Printer: Provides line spacing or a skip to a specific carriage control channel. You cannot use a CNTRL macro if carriage control characters are provided in the record. If the printer contains the universal character set feature, data checks should be blocked (OPTCD=U should not appear in the data control block).

Magnetic Tape: Provides method of forward spacing and backspacing (BSAM only for a data set not open for output). If OPTCD=H is indicated in the data control block, you can use the CNTRL macro to perform record positioning on VSE [2] tapes that contain embedded checkpoint records. Embedded checkpoint records found during the record positioning are bypassed and are not counted as blocks spaced over. OPTCD=H must be specified in a job control language DD statement. The CNTRL macro cannot be used to backspace VSE 7-track tapes written in data convert mode that contain embedded checkpoint records (BSAM).

Do not use the CNTRL macro with output operations on BSAM tape data sets.

3525 Printing: Provides line spacing or a skip to a specific printing line on the card. The card contains 25 printing lines; the odd-numbered lines 1 through 23 correspond to the printer skip channels 1 through 12 (see the SK parameter).

The CNTRL macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The format of the CNTRL macro is:

[*label*]	CNTRL	*dcb address* {,SS,{1|2}} {,SP,{1|2|3}} {,SK,{1|2|...|11|12}} {,BSM} {,FSM} {,BSR[,*number of blocks*]} {,FSR[,*number of blocks*]}

[2] VSE (Virtual Storage Extended) tapes used to be called DOS tapes.

dcb address—RX-Type Address or (2-12)
specifies that the address of the data control block for the data set that is opened for the online device. When issued in 31-bit addressing mode, the input DCB address must be a clean 31-bit address.

SS,{1|2}
specifies that the control function that is requested is stacker selection on a card reader. Either 1 or 2 must be coded to indicate which stacker is selected.

SP,{1|2|3}
specifies that the control function that is requested is printer line spacing or 3525 card punch line spacing. Either 1, 2, or 3 must be coded to indicate the number of spaces for each print line.

SK,{1|2|...|11|12}
specifies that the control function that is requested is a skip operation on the printer or 3525 card punch, print feature. A number (1 through 12) must be coded to indicate the channel or print line to which the skip is to be taken.

BSM
specifies that the control function that is requested is to backspace the magnetic tape past a tape mark, then forward space over the tape mark.

FSM
specifies that the control function that is requested is to forward space the magnetic tape over a tape mark, then backspace past the tape mark.

BSR
specifies that the control function that is requested is to backspace the magnetic tape the number of blocks indicated in *number-of-blocks*.

FSR
specifies that the control function that is requested is to forward space the magnetic tape the number of blocks indicated in *number-of-blocks*.

***number of blocks*—symbol, decimal digit, absexp, or (2-12)**
specifies the number of blocks to backspace (see BSR parameter) or forward space (see FSR parameter) the magnetic tape. The maximum value that can be specified is 32767. If *number-of-blocks* is omitted, 1 is assumed.

If the forward space or backspace operation is not completed successfully, control is passed to the error analysis (SYNAD) routine. If no SYNAD exit routine is designated, the task is abnormally terminated.

For more information on register contents when control is passed to the error analysis routine, see *z/OS DFSMS Using Data Sets*. If a tape mark is found for BSR or FSR, control is returned to the processing program, and register 15 contains a count of the uncompleted forward spaces or backspaces. If the operation is completed normally, register 15 contains the value zero. If CNTRL encounters a tape mark, it moves the tape back over the tape mark before returning to the user.

DCB—Construct a data control block (BDAM)

The data control block for a basic direct access method (BDAM) data set is constructed during assembly of the problem program. You must code DSORG and MACRF in the DCB macro, but the other parameters can be supplied to the DCB from the DD statement or an existing data set label (DSCB). If more than one of these sources specifies information for a particular field, the order of priority is the DCB macro, DD statement, and data set label. Each BDAM DCB parameter description contains a heading, "Source". The information under this heading describes the sources that can supply the parameter.

Each reference to a DCB OPEN exit routine applies also to a JFCBE exit routine.

You can assemble the DCB macro into a program that resides above the 16MB line, but the program must move it below the line before using it. All areas that the DCB refers to, such as EXLST and SYNAD, must be below the 16MB line.

The format of the DCB macro for BDAM is:

[*label*]	DCB	[BFALN={F|D}] [,BFTEK=R] [,BLKSIZE=*absexp*] [,BUFCB=*relexp*] [,BUFL=*absexp*] [,BUFNO=*absexp*] [,DCBE=*relexp*] [,DDNAME=*symbol*] "1" on page 172 ,DSORG={DA|DAU} [,EXLST=*relexp*] [,KEYLEN=*absexp*] [,LIMCT=*absexp*] ,MACRF={{(R{K[I]|I}[X][S][C])} {(W{A[K][I]|K[I]|I}[C])} {(R{K[I]|I}[X] [S][C] ,W{A[K][I]|K [I]|I}[C])}} [,OPTCD={[R|A][E][F][W]}] [,RECFM={U|V[S|BS]|F[T]}] [,SYNAD=*relexp*]

Note:

1. This parameter must be supplied before an OPEN macro is issued for this DCB; it cannot be supplied in the open exit routine.

Recommendation: When creating a DCB to open a data set allocated to an SMS-managed volume, do not specify values that would change the data set to a type which cannot be SMS-managed, such as DSORG=DAU.

BDAM supports the following DCB parameters:

BFALN={F|D}
specifies the boundary alignment for each buffer in the buffer pool. You can specify the BFALN parameter when (1) BSAM is being used to allocate a direct data set and buffers are acquired automatically, (2) when an existing BDAM data set is being processed and dynamic buffering is requested, or (3) when the GETPOOL macro is used to construct the buffer pool. If BFALN is omitted, the system provides doubleword alignment for each buffer. You can specify:

F
specifies that each buffer is on a fullword boundary that is not also a doubleword boundary.

D
specifies that each buffer is on a doubleword boundary.

If you use the BUILD macro to construct the buffer pool, or if the problem program controls all buffering, the problem program must provide the area for the buffers and control buffer alignment.

Source: BFALN can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. If both BFALN and BFTEK are specified, they must be supplied from the same source.

BFTEK=R
specifies that the data set is allocated for or contains variable-length spanned records. You can code BFTEK=R only when the record format is specified as RECFM=VS.

When variable-length spanned records are written, the data length can exceed the total capacity of a single track on the direct access storage device being used, or it can exceed the remaining capacity on a given track. The system divides the data block into segments (if necessary), writes the first segment on a track, and writes the remaining segments on the following track(s).

When a variable-length spanned record is read, the system reads each segment and assembles a complete data block in the buffer designated in the *area address* of a READ macro.

Variable-length spanned records can also be read using BSAM. When BSAM is used to read a BDAM variable-length spanned record, the record is read one segment at a time, and the problem program must assemble the segments into a complete data block. This operation is described in the section for the BSAM DCB macro.

Source: BFTEK can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. If both BFTEK and BFALN are specified, they must be supplied from the same source.

BLKSIZE=*absexp* (maximum value is 32760)
specifies the length, in bytes, of each data block for fixed-length records. Or, specifies the maximum length, in bytes, of each data block for variable-length or undefined-length records. If keys are used, the length of the key is not included in the value specified for BLKSIZE.

The actual value that you can specify in BLKSIZE depends on the record format and the type of direct access storage devices being used. If variable-length spanned records are used, the value specified in BLKSIZE can be up to the maximum. For all other record formats (F, V, VBS, and U), the maximum value that can be specified in BLKSIZE is determined by the track capacity of a single track on the direct access storage device being used. Device capacity for direct access storage devices is described in Appendix E, “Selecting logical record lengths and block sizes for specific devices,” on page 403. For additional information about space allocation, see *z/OS DFSMS Using Data Sets*.

Source: BLKSIZE can be supplied in the DCB macro, in the DCB subparameter of a DD statement, by the problem program before completion of the data control block exit routine, or by the data set label of an existing data set. Block size can also be derived from the JCL keyword LIKE. See *z/OS MVS JCL Reference* and *z/OS MVS JCL User's Guide* for more information on LIKE.

BUFCB=*relexp*
specifies the address of the buffer pool control block in a buffer pool constructed by a BUILD macro. The buffer pool must reside below the 16MB line.

If the buffer pool is constructed automatically, dynamically, or by a GETPOOL macro, you do not need to use BUFCB because the system places the address of the buffer pool control block into the data control block. Also, if the problem program is to control all buffering, omit BUFCB.

Source: BUFCB can be supplied in the DCB macro or by the problem program before completion of the data control block exit routine. If the problem program is to control all buffering (and BUFNO is not supplied by any source), then BUFCB can be supplied any time before it is needed. You do not have to have a buffer pool.

BUFL=*absexp* (maximum value KEYLEN + BLKSIZE is 32760)
specifies the length, in bytes, of each buffer in the buffer pool when the buffers are acquired automatically (create BDAM) or dynamically (existing BDAM).

When buffers are acquired automatically (create BDAM), the BUFL parameter is optional. If specified, the value must be at least as large as the sum of the values specified for KEYLEN and BLKSIZE. If BUFL is omitted, the system builds buffers with a length equal to the sum of the values specified in KEYLEN and BLKSIZE.

You must specify BUFL when processing an existing direct data set with dynamic buffering. Its value must be at least as large as the value specified for BLKSIZE when the READ or WRITE macro specifies a key address, or the value specified in BUFL must be at least as large as the sum of the values specified in KEYLEN and BLKSIZE if the READ and WRITE macros specify `'S'` for the key address.

You can omit BUFL if the buffer pool is constructed by a BUILD or GETPOOL macro, or if the problem program controls all buffering.

Source: BUFL can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

BUFNO=*absexp* (maximum value is 255)
specifies the number of buffers to be constructed by a BUILD macro, or the number of buffers and segment work areas to be acquired automatically by the system.

If the buffer pool is constructed by a BUILD macro or if buffers are acquired automatically when BSAM is used to allocate a direct data set, you must specify the number of buffers in BUFNO.

If dynamic buffering is requested when an existing direct data set is being processed, BUFNO is optional; if omitted, the system acquires two buffers.

If variable-length spanned records are being processed and dynamic buffering is requested, the system also acquires a segment work area for each buffer. If dynamic buffering is not requested, the system acquires the number of segment work areas specified in BUFNO. If BUFNO is omitted when variable-length spanned records are being processed and dynamic buffering is not requested, the system acquires two segment work areas.

If the buffer pool is constructed by a GETPOOL macro or if the problem program controls all buffering, you can omit BUFNO unless you need it to acquire additional segment work areas for variable-length spanned records.

Source: BUFNO can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

DCBE=*relexp*
specifies the address of a DCB Extension (DCBE). The DCBE may reside above the 16MB line. You may assemble a DCB and DCBE in a program that resides above the line if the DCB is copied below the line before opening the copy.

If the DCBE is specified, it must be specified before issuing the OPEN macro. Like the DCB, the DCBE must exist until the data set is closed. Otherwise, there may be unpredictable results.

Only one open DCB at a time can refer to a particular DCBE. After a DCB is successfully closed, you can open a different DCB referring to the DCBE.

The DCBE is not required with BDAM unless the data set requires a DCBE option or if you choose to use DCBE options.

If a DCBE exists, the flags DCBH0 and DCBH1 are both set on. The pointer to the DCBE is stored at offset +0 in the DCB.

Source: You can supply the DCBE address in the DCB macro or before issuing an OPEN macro to open the data set.

DDNAME=*symbol*
specifies the name used to identify the job control language data definition (DD) statement that defines the data set being allocated or processed.

Source: DDNAME can be supplied in the DCB macro or can be moved into the DCB by the problem program before an OPEN macro is issued to open the data set.

DSORG={DA|DAU}
specifies the data set organization and whether the data set contains any location-dependent information that would make it unmovable. For example, if actual device addresses are used to process a BDAM data set, the data set can be unmovable. You can specify:

DA
specifies a direct organization data set.

DAU
specifies a direct organization data set containing location-dependent information that would make it unmovable.

Restriction:

A DSORG=DAU data set cannot be SMS-managed.

When a direct data set is allocated, the basic sequential access method (BSAM) is used. You must code DSORG in the DCB macro as DSORG=PS or PSU when the data set is allocated, and code the DCB subparameter in the corresponding DD statement as DSORG=DA or DAU. This creates a data set with a data set label identifying it as a direct data set.

Source: DSORG must be specified in the DCB macro. See the preceding comment about creating a direct data set.

EXLST=*relexp*
specifies the address of the DCB exit list. The EXLST parameter is required if the problem program processes user labels during the open or close routine, if the data control block exit routine is used for additional processing, or if the DCB ABEND exit is used for abend condition analysis.

The exit list must reside below the line. For the functions, format, and requirements of exit list processing, see *z/OS DFSMS Using Data Sets*. Exit routines can reside above the 16 MB line if you use the technique described in Figure 4 on page 153.

Source: EXLST can be supplied in the DCB macro or by the problem program before the relevant function is needed.

KEYLEN=*absexp* (maximum value is 255)
specifies the length, in bytes, of all keys used in the data set. When keys are used, a key is associated with each data block in the data set. If the key length is not supplied by any source, no input or output requests that require a key can be specified in a READ or WRITE macro.

Source: KEYLEN can be supplied in the DCB macro, in the DCB subparameter of a DD statement, by the problem program before the completion of the data control block exit routine, or by an existing data set label. If KEYLEN=0 is specified in the DCB macro, a special indicator is set in RECFM so that KEYLEN cannot be supplied from the DCB subparameter of a DD statement or data set label of an existing data set. KEYLEN=0 can be coded only in the DCB macro and will be ignored if specified in the DD statement.

Key length can be derived from the data class associated with the data set. Key length can also be derived from the JCL keyword LIKE. However, if KEYLEN is specified in the DCB macro, it overrides the value derived from data class or LIKE. For more information, see *z/OS MVS JCL Reference*.

LIMCT=*absexp*
specifies the number of blocks or tracks to be searched when the extended search option (OPTCD=E) is requested.

When the extended search option is requested and relative block addressing is used, the records must be fixed-length record format. The system converts the number of blocks specified in LIMCT into the number of tracks required to contain the blocks, then proceeds in the manner described below for relative track addressing.

When the extended search option is requested and relative track addressing is used (or the number of blocks has been converted to the number of tracks), the system searches for two things: (1) the block specified in a READ or WRITE macro (type DK), or (2) available space where it can add a block (WRITE macro, type DA). The search is as follows:

1. The search begins at the track specified by the *block address* of a READ or WRITE macro.
2. The search continues until the search is satisfied, the number of tracks specified in LIMCT have been searched, or the entire data set has been searched. If the search is not satisfied when the last track of the data set is reached, the system continues the search by starting at the first track of the data set if the EOF marker is on the last track allocated to the data set. (This operation allows the number specified in LIMCT to exceed the size of the data set, causing the entire data set to be searched.) You can ensure that the EOF marker is on the last allocated track by determining the size of the data set and allocating space in blocks, or by allocating space in tracks and including the RLSE subparameter in the SPACE parameter of the DD statement (RLSE specifies that all unused tracks be returned to the system).

The problem program can change the DCBLIMCT field in the data control block at any time, but, if the extended search option is used, the DCBLIMCT field must not be zero when a READ or WRITE macro is issued.

If the extended search option is not requested, the system ignores LIMCT, and the search for a data block is limited to a single track.

Source: LIMCT can be supplied in the DCB macro, the DCB subparameter of a DD statement, or by the problem program before the count is required by a READ or WRITE macro.

MACRF={{(R{K[I]|I}[X][S][C])}
{(W{A[K][I]|K [I]|I}[C])}
{(R{K[I]|I}[X] [S][C],W{A[K][I] |K[I]|I}[C])}}

specifies the type of macros (READ, WRITE, CHECK, and WAIT) that are used to process the data set. MACRF also specifies the type of search argument and BDAM functions that are used with the data set. When BSAM is used to create a direct data set, the BSAM parameter MACRF=WL is specified. This special parameter invokes the BSAM routine that can create a BDAM data set. You can specify the following characters for BDAM:

A
specifies that data blocks are added to the data set.

C
specifies that the CHECK macro is used to test for completion of read and write operations. If C is not specified, WAIT macros must be used to test for completion of read and write operations.

I
specifies that the search argument is the block identification portion of the data block. If relative addressing is used, the system converts the relative address to an actual address (MBBCCHHR) before the search.

K
specifies that the search argument is the key portion of the data block. The location of the key to be used as a search argument is specified in a READ or WRITE macro.

R
specifies that READ macros are used. READ macros can be issued when the data set is opened for INPUT, OUTPUT, or UPDAT. R is required if the OPEN option is INPUT or UPDAT. It has no effect if the OPEN option is OUTPUT or EXTEND.

S
specifies that dynamic buffering is requested by specifying 'S' in the area address parameter of a READ or WRITE macro.

W
specifies that WRITE macros are used. WRITE macros can be issued only when the data set is opened for OUTPUT or UPDAT. W is required if the OPEN option is OUTPUT. It has no effect if the OPEN option is INPUT.

X
specifies that READ macros request exclusive control of a data block. When exclusive control is requested, the data block must be released by a subsequent WRITE or RELEX macro.

Source: MACRF must be supplied in the DCB macro.

OPTCD={[R|A][E][F][W]}

specifies the optional services used with the direct data set. These options are related to the type of addressing used, the extended search option, block position feedback, and write-validity checking. You can code the following characters in any order, in any combination, and without commas between characters:

A
specifies that actual device addresses (MBBCCHHR) are provided to the system when READ or WRITE macros are issued.

E
specifies that the extended search option is used to locate data blocks or available space where a data block can be added. When the extended search option is specified, the number of blocks or tracks to be searched must be specified in LIMCT. The extended search option is ignored if actual addressing (OPTCD=A) is also specified. The extended search option requires that the data set have keys and that the search be made by key (by specifying DK in the READ or WRITE macro or DA in the WRITE macro).

F
specifies that the block position feedback that is requested by a READ or WRITE macro is to be in the same form originally presented to the system in the READ or WRITE macro. If the F parameter is omitted, the system provides feedback, when requested, as an 8-byte actual device address. (Feedback is always provided if exclusive control is requested.)

R
specifies that relative block addresses (as 3-byte binary numbers) are provided to the system when a READ or WRITE macro is issued.

W
specifies that the system is to perform a validity check for each record written.

Tip: You can specify relative track addressing by omitting both A and R from OPTCD. If you want to specify relative track addressing after your data set has been accessed using another addressing scheme (OPTCD=A or OPTCD=R), you should either specify a valid OPTCD subparameter (`E, F,` or `W`) in the DCB macro or DD statement when you reopen your data set, or zero out the OPTCD=A or OPTCD=R bits in the data control block exit routine. Note that the first method prevents the open routines from merging any of the other OPTCD bits from the format-1 DSCB in the DCB. Both methods update the OPTCD bits in the DSCB if the open is for OUTPUT, OUTIN, or UPDAT.

Source: OPTCD can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the DCB open exit routine.

RECFM={U|V[S|BS]|F[T]}
specifies the record format and characteristics of the data set being allocated or processed. You can specify the following characters. (If the optional characters are coded, they must be coded in the order shown above).

B
specifies that the data set contains blocked records. The record format RECFM=VBS is the only combination in which B can be specified. RECFM=VBS does not cause the system to process spanned records. The problem program must block and segment the records. RECFM=VBS is treated as a variable-length record by BDAM.

F
specifies that the data set contains fixed-length records.

S
specifies that the data set contains variable-length spanned records when it is coded as RECFM=VS. When RECFM=VBS is coded, the records are treated as variable-length records, and the problem program must block and segment the records.

T
specifies that track overflow is used with the data set. Track overflow allows a record to be partially written on one track and the remainder is written on the following track (if required).

Note: This is an obsolete option. The system ignores it.

U
specifies that the data set contains undefined-length records.

V
specifies that the data set contains variable-length records.

Source: RECFM can be supplied in the DCB macro, in the DCB subparameter of a DD statement, the problem program before completion of the data control block exit routine, or by the data set label of an existing data set.

Record format can be derived from the data class associated with the data set. Record format can also be derived from the JCL keyword LIKE. However, if RECFM is specified in the DCB macro, it overrides the value derived from data class or LIKE. For more information, see *z/OS MVS JCL Reference*.

SYNAD=*relexp*
specifies the address of the error analysis routine to be given control when an uncorrectable input/output error occurs. The entry point of this SYNAD routine must reside below the line. The entry point of this SYNAD routine must reside below the line. The contents of the registers when the error analysis routine is given control are described in *z/OS DFSMS Using Data Sets*. Additional status information available to the SYNAD routine is described in "Status information following an input/output operation" on page 373.

The error analysis routine must not use the save area pointed to by register 13. The system does not restore registers when it regains control from the error analysis routine. The error analysis routine can issue a RETURN macro that uses the address in register 14 to return control to the system. When control is returned in this manner, the system returns control to the problem program and proceeds as though no error had been found. When a direct data set is being created, a return from the error analysis routine to the system causes abnormal end of the task.

When you issue a CHECK macro, the SYNAD routine receives control if an I/O error occurred. If SYNAD is omitted, the task is abnormally terminated if you issue a CHECK macro and it finds an uncorrectable I/O error.

SYNAD receives control in the addressing mode in which the CHECK macro was issued. On return from a SYNADAF or SYNADRLS macro issued in the SYNAD routine, the high order byte of register 15 will be unpredictable. Therefore, callers of SYNADAF or SYNADRLS in 31-bit addressing mode must either not use register 15 as a base register or restore the high order bytes on return from SYNADAF or SYNADRLS.

Source: SYNAD can be supplied in the DCB macro or by the problem program. The problem program can also change the error routine address at any time.

DCB—Construct a data control block (BISAM)

Recommendation: The system no longer supports indexed sequential data sets. Convert the data set to a key sequenced data set (KSDS) and use the ISAM interface of VSAM or convert your program to use VSAM.

The data control block for a basic indexed sequential access method (BISAM) data set is constructed during assembly of the problem program. You must code DSORG and MACRF in the DCB macro, but the other DCB parameters can be supplied to the data control block from other sources. Each BISAM DCB parameter description contains a heading, "Source". The information under this heading describes the sources that can supply the parameters. Each reference to a DCB OPEN exit routine applies also to a JFCBE exit routine.

You can assemble the DCB macro into a program that resides above the 16MB line, but the program must move it below the line before using it.

The format of the DCB macro for BISAM is:

[*label*]	DCB	[BFALN={F|D}] [,BUFCB=*relexp*] [,BUFL=*absexp*] [,BUFNO=*absexp*] [,DDNAME=*symbol*] "1" on page 179 ,DSORG=IS [,EXLST=*relexp*] ,MACRF={{(R[S][C])} {(W{U[A]|A}[C])} {(R[U[S]|S][C],W{U [A]|A}[C])}} [,MSHI=*relexp*] [,MSWA=*relexp*] [,NCP=*absexp*] [,OPTCD={([L][R][W])}] [,SMSI=*absexp*] [,SMSW=*absexp*] [,SYNAD=*relexp*]

Note:

1. This parameter must be supplied before an OPEN macro is issued for this DCB; it cannot be supplied in the open exit routine.

BISAM supports the following DCB parameters:

BFALN={F|D}
specifies the boundary alignment for each buffer in the buffer pool when the buffer pool is acquired for use with dynamic buffering or when the buffer pool is constructed by a GETPOOL macro. If BFALN is omitted, the system provides doubleword alignment for each buffer. You can specify:

F
specifies that each buffer is on a fullword boundary that is not also a doubleword boundary.

D
specifies that each buffer is on a doubleword boundary.

If the BUILD macro is used to construct the buffer pool, or if the problem program controls all buffering, the problem program must provide an area for the buffers and control buffer alignment.

Source: BFALN can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

BUFCB=*relexp*
specifies the address of the buffer pool control block when the buffer pool is constructed by a BUILD macro.

You can omit BUFCB if you request dynamic buffering or use the GETPOOL macro to construct the buffer pool, because the system places the address of the buffer pool control block into the data control block. Also, if the problem program is to control all buffering, omit BUFCB.

Source: BUFCB can be supplied in the DCB macro or by the problem program before completion of the data control block exit routine.

BUFL=*absexp* (maximum value is 32760)
specifies the length, in bytes, of each buffer in the buffer pool to be constructed by a BUILD or GETPOOL macro. When the data set is opened, the system computes the minimum buffer length required and verifies that the length in the buffer pool control block is equal to or greater than the minimum length required. The system then inserts the computed length into the BUFL field of the data control block.

If dynamic buffering is requested, the system computes the buffer length required, and BUFL is not required.

If the problem program controls all buffering, BUFL is not required. However, an indexed sequential data set requires additional buffer space for system use. For a description of the buffer length required for various ISAM operations, see *z/OS DFSMS Using Data Sets*.

Source: BUFL can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

BUFNO=*absexp* (maximum value is 255)
specifies the number of buffers that are requested for use with dynamic buffering. If dynamic buffering is requested but BUFNO is omitted, OPEN automatically acquires two buffers for use with dynamic buffering.

If the GETPOOL macro is used to construct the buffer pool, BUFNO is not required.

Source: BUFNO can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

DDNAME=*symbol*
specifies the name used to identify the job control language data definition (DD) statement that defines the indexed sequential data set being allocated or processed.

Source: DDNAME can be supplied in the DCB macro or by the problem program before an OPEN macro is issued to open the data set.

DSORG=IS
specifies the indexed sequential organization of the data set. IS is the only combination of characters that can be coded for BISAM.

Source: Unless it is for a data set passed from a previous job step, DSORG must be coded in the DCB macro and in the DCB subparameter of a DD statement. In this case, DSORG can be omitted from the DD statement.

EXLST=*relexp*
specifies the address of the DCB exit list. EXLST is required only if the problem program uses the data control block OPEN exit routine for additional processing.

For the functions, format, and requirements for exit list processing, see *z/OS DFSMS Using Data Sets*. The exit list must reside below the line.

Source: EXLST can be supplied in the DCB macro or by the problem program before the relevant function is needed.

MACRF={{(R[S][C])}
{(W{U[A]|A}[C])}
{(R[U[S]|S][C], W{U[A]|A}[C])}}
specifies the type of macros (READ, WRITE, CHECK, WAIT, and FREEDBUF) and type of processing (add records, dynamic buffering, and update records) to be used with the data set being processed. You can code the parameter in any of the combinations shown above. The following characters can be coded for BISAM:

A
specifies that new records are to be added to the data set. This character must be coded if WRITE KN macros are used with the data set.

C
specifies that the CHECK macro is used to test I/O operations for completion. If C is not specified, WAIT macros must be used to test for completion of I/O operations.

R
specifies that READ macros are to be used. R is required if the OPEN option is INPUT or UPDAT. It has no effect if the OPEN option is OUTPUT or EXTEND.

S
specifies that dynamic buffering is requested in READ macros. Do not specify S if the problem program provides the buffer pool.

U
specifies that records in the data set are to be updated in place. If U is coded in combination with R, it must also be coded in combination with W. For example, MACRF=(RU,WU).

W
specifies that WRITE macros are to be used. W is required if the OPEN option is OUTPUT. It has no effect if the OPEN option is INPUT.

Source: MACRF must be coded in the DCB macro.

MSHI=*relexp*
specifies the address of the storage area that are used to contain the highest-level master index for the data set. The system uses this area to reduce the search time required to find a given record in the data set. MSHI is coded only when SMSI is coded.

Source: MSHI can be supplied in the DCB macro or by the problem program before completion of the data control block exit routine.

MSWA=*relexp*
specifies the address of the storage work area to be used by the system when new records are being added to the data set. This parameter is optional, but the system acquires a minimum-size work area if the parameter is omitted. MSWA is coded only when the SMSW parameter is coded.

Processing efficiency can be increased if more than a minimum-size work area is provided. For more detailed information about work area size, see *z/OS DFSMS Using Data Sets*.

Source: MSWA can be supplied in the DCB macro or by the problem program before completion of the data control block exit routine.

NCP=*absexp* (maximum value is 99)
specifies the maximum number of READ and WRITE macros issued before the first CHECK (or WAIT) macro is issued to test for completion of the I/O operation. The maximum number can be less than 99, depending on the amount of virtual storage available below the line in the address space. If NCP is omitted, 1 is assumed. If dynamic buffering is used, the value specified for NCP must not exceed the number of buffers specified in BUFNO.

Source: NCP can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block open exit routine.

OPTCD=([L][R][W])
specifies the optional services that are performed by the control program when creating or updating an indexed sequential data set. You must request all optional services by one method. That is, by the data set label of an existing data set, this macro, or the DD statement on the DCB parameter. However, it can be modified by the problem program. You can code the following characters in any order, in any combination, and without commas between characters:

L
specifies that the control program delete records that have a first byte of X'FF'. (These records can be deleted when space is required for new records. To use the delete option, the relative key position (RKP) must be greater than 0 for fixed-length records and greater than 4 for variable-length records.)

R
specifies that the control program place reorganization statistics in certain fields of the data control block. The problem program can analyze these statistics to determine when to reorganize the data set. If OPTCD is omitted, the reorganization statistics are automatically provided. However, if you use OPTCD, you must specify OPTCD=R to get the reorganization statistics.

W
specifies a validity check for write operations on direct access storage devices.

SMSI=*absexp* (maximum value is 65535)
specifies the length, in bytes, that is required to contain the highest-level master index for the data set being processed. Look at the DCBNCRHI field of the data control block to determine the size that is required. When an indexed sequential data set is created (with QISAM), the size of the highest-level index is inserted into the DCBNCRHI field. If the value that is specified in SMSI is less than the value in the DCBNCRHI field, the task is abnormally terminated.

Source: SMSI can be supplied in the DCB macro or by the problem program before completion of the data control block exit routine.

SMSW=*absexp* (maximum value is 65535)
specifies the length, in bytes, of a work area that is used by BISAM. This parameter is optional, but the system acquires a minimum-size work area if the parameter is omitted. Code SMSW together with MSWA. If you code SMSW but the size you specify is less than the minimum that is required, the task is abnormally terminated. *z/OS DFSMS Using Data Sets* describes the methods of calculating the size of the work area.

If unblocked records are used, the work area must be large enough to contain all the count fields (8 bytes each), key fields, and data fields that are contained on one direct access storage device track.

If blocked records are used, the work area must be large enough to contain all the count fields (8 bytes each) and data fields that are contained on one direct access storage device track plus additional space for one logical record (LRECL value).

Source: SMSW can be supplied in the DCB macro or by the problem program before completion of the data control block exit routine.

SYNAD=*relexp*
specifies the address of the error analysis routine given control when an uncorrectable input/output error occurs. The entry point of this SYNAD routine must reside below the line. The contents of the registers when the error analysis routine is given control are described in *z/OS DFSMS Using Data Sets*. Additional status information available to the SYNAD routine is described in “Status information following an input/output operation” on page 373.

The error analysis routine must not use the save area pointed to by register 13. The system does not restore registers when it regains control from the error analysis routine. The error analysis routine can issue a RETURN macro that uses the address in register 14 to return control to the system. When control is returned in this manner, the system returns control to the problem program and proceeds as though no error had been found. If the error analysis routine continues processing, the results are unpredictable.

When you have issued the CHECK macro, the SYNAD routine receives control if an I/O error occurs. If SYNAD is omitted, the task is abnormally terminated when an uncorrectable input/output error occurs.

Source: SYNAD can be supplied in the DCB macro or by the problem program. The problem program can also change the error analysis routine address at any time.

DCB—Construct a data control block (BPAM)

The data control block for a basic partitioned access method (BPAM) data set is constructed during assembly of the problem program. You must code the DSORG and MACRF parameters in the DCB macro, but the other DCB parameters can be supplied from other sources. Each of the BPAM DCB parameter descriptions contains a heading, "Source". The information under this heading describes the sources that can supply the parameter to the data control block. Each reference to a DCB OPEN exit routine applies also to a JFCBE exit routine. The DCB fields that you can test or set are described in Appendix B, “Non-VSAM control blocks,” on page 373.

You can assemble the DCB macro into a program that resides above the 16 MB line, but the program must move it below the line before using it. Except for the DCBE, all areas that the DCB refers to, such as EXLST and EODAD, must be below the 16 MB line.

The format of the DCB macro for BPAM is:

[*label*]	DCB	[BFALN={F|D}] [,BLKSIZE=*absexp*] [,BUFCB=*relexp*] [,BUFL=*absexp*] [,BUFNO=*absexp*] [,DCBE=*relexp*] "1" on page 183 [,DDNAME=*symbol*] "1" on page 183 ,DSORG={PO|POU} [,EODAD=*relexp*] [,EXLST=*relexp*] [,KEYLEN=*absexp*] [,LRECL=*absexp*] ,MACRF={(R|W|R,W)} "1" on page 183 [,NCP=*absexp*] [,OPTCD={C|W[C]} [,RECFM={{U[T][A|M]} {V[B[T]|T][A|M]} {F[B[T]|T][A|M]}}] [,SYNAD=*relexp*]

Note:

1. This parameter must be supplied before an OPEN macro is issued for this DCB; it cannot be supplied in the open exit routine.

Note: When creating a DCB to open a data set allocated to an SMS-managed volume, do not specify values that would change the data set to a type which cannot be SMS-managed, such as DSORG=POU. Refer to *z/OS DFSMS Using Data Sets* for further information.

When you create or process a partitioned data set or PDSE, you can specify the following parameters in the DCB macro:

BFALN={F|D}
specifies the boundary alignment for each buffer in the buffer pool when the buffer pool is constructed automatically or by a GETPOOL macro. If BFALN is omitted, the system provides doubleword alignment for each buffer. You can specify the following characters in BFALN:

F
specifies that each buffer is aligned on a fullword boundary that is not also a doubleword boundary.

D
specifies that each buffer is aligned on a doubleword boundary.

If the BUILD macro is used to construct the buffer pool or if the problem program controls all buffering, the problem program must provide an area for the buffers and control buffer alignment.

Source: BFALN can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

BLKSIZE=*absexp* (maximum value KEYLEN + BLKSIZE is 32760)
specifies the length, in bytes, of each data block for fixed-length unblocked records. Or, it specifies the maximum length, in bytes, for any other record format. If keys are used, the length of the key is not included in the value specified for BLKSIZE.

You can request to use the BLKSIZE keyword on a DCBE macro. This is the large block interface (LBI). If the system allows the LBI, the system modifies the BLKSIZE field in the DCB and your program should not use it.

The actual block size you can specify depends on the record format and type of direct access storage devices being used. The block size can be up to 32760 but you will get more data on each track if you write shorter blocks. Device capacity for direct access storage devices is described in Appendix

E, “Selecting logical record lengths and block sizes for specific devices,” on page 403. For additional information about space allocation, see *z/OS DFSMS Using Data Sets*.

For fixed-length records, the value specified in BLKSIZE should be a multiple of the value specified for the logical record length (LRECL).

For fixed-length unblocked records, LRECL must equal BLKSIZE (if LRECL is specified).

For variable-length records, the value specified in BLKSIZE must include the maximum logical record length (up to 32756 bytes) plus 4 bytes for the block descriptor word (BDW).

For undefined-length records, the value specified for BLKSIZE can be altered by the problem program when the actual length becomes known to the problem program. The value can be inserted into the DCBBLKSI field of the data control block, or DCBEBLKSI field of the DCBE, or specified in the *length* parameter of a READ or WRITE macro.

Processing PDSEs: The system reblocks PDSE records into its own internal format when the data set is written, and reconstructs the blocks using the block size from the DCB when the data set is read. For fixed-length blocked records, the value specified in BLKSIZE *must* be a multiple of the value in LRECL (if LRECL is specified). The LRECL value must be available to OPEN when the PDSE is open for output.

When reading a PDSE directory using fixed-length blocked records, you can specify a BLKSIZE of 256 or greater (the LRECL is ignored).

System-Determined Block Size: IBM recommends that you not specify block size unless the record format is U. This makes your program less dependent on the physical characteristics of the device although a PDSE block size has little to do with device characteristics. If the block size is not specified when the data set is allocated, and the LRECL and RECFM are known, the system derives an optimum block size for the data set. This system-determined block size is retained in the data set label. When the data set is opened for output, OPEN checks the block size in the data set label. If it is a system-determined block size, and the LRECL or RECFM have changed from those specified in the data set label, OPEN redetermines an optimum block size for the data set.

Source: BLKSIZE can be supplied in the DCB or DCBE macro, in the DCB subparameter of a DD statement, by the problem program before completion of the data control block exit routine, by the data set label of an existing data set, or by the system determining a value for a new data set. The system does not copy BLKSIZE when you code the JCL keyword LIKE. It derives the BLKSIZE from RECFM and LRECL which can be copied. For more information on LIKE, see *z/OS MVS JCL Reference* and *z/OS MVS JCL User's Guide*.

BUFCB=*relexp*
specifies the address of the buffer pool control block that you have constructed by a BUILD macro.

If the buffer pool is constructed automatically or by a GETPOOL macro, you can omit the BUFCB parameter because the system places the address of the buffer pool control block into the data control block. Also, if the problem program is to control all buffering, omit the BUFCB parameter. A buffer pool control block resides below the 16MB line.

Source: BUFCB can be supplied in the DCB macro or by the problem program before issuing a GETBUF macro.

BUFL=*absexp* (maximum value is 32760)
specifies the length, in bytes, of each buffer in the buffer pool when the buffer pool is acquired automatically. If BUFL is omitted and you request OPEN to build a buffer pool, the system acquires buffers with a length equal to the sum of the values specified in KEYLEN and BLKSIZE. If the problem program requires longer buffers (up to 32760 bytes), specify BUFL.

If the problem program controls all buffering, BUFL is not required.

Source: BUFL can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

BUFNO=*absexp* (maximum value is 255)
specifies the number of buffers to be constructed by a BUILD macro. Or, it specifies the number of buffers to be acquired automatically by the system.

If the problem program controls all buffering or if the buffer pool is constructed by a GETPOOL macro, omit BUFNO.

The default value is zero. If the blocksize is less than 32768 and BUFNO is either specified as zero to allowed to default to zero the system does not acquire buffers automatically. If the blocksize is 32768 or greater and BUFNO is either specified as zero or allowed to default to zero then the system will acquire two buffers or the number of buffers specified by MULTSDN, whichever is greater. If the system acquires buffers for BPAM, they reside below the 16MB line. You may obtain each buffer by issuing a GETBUF macro.

Source: BUFNO can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

DCBE=*relexp*
specifies the address of a DCB extension (DCBE). The DCBE may reside above the 16 MB line. You may assemble a DCB and DCBE in a program that resides above the line if the DCB is copied below the line before opening the copy.

If the DCBE is specified, it must be specified before issuing the OPEN macro. Like the DCB, the DCBE must exist until the data set is closed. Otherwise, there may be unpredictable results.

Only one open DCB at a time can refer to a particular DCBE. After a DCB is successfully closed, you can open a different DCB that refers to the DCBE.

The DCBE is not required with BPAM unless the data set requires a DCBE option or if you choose to use DCBE options.

If a DCB points to a DCBE, the flags DCBH0 and DCBH1 are both set on. The pointer to the DCBE is stored at offset +0 in the DCB (and replaces the field DCBRELAD). If a DCBE exists, data that would be stored at DCBRELAD is stored in the DCBE (DCBERELA). If a DCBE does not exist, DCBRELAD continues to be located at offset +0 in the DCB.

Source: You can supply the DCBE address in the DCB macro or before issuing an OPEN macro to open the data set.

DDNAME=*symbol*
specifies the name that is used to identify the job control language data definition (DD) statement that defines the data set being allocated or processed.

Source: DDNAME can be supplied in the DCB macro or by the problem program before an OPEN macro is issued to open the data set.

DSORG={PO|POU}
specifies the data set organization and whether the data set contains any location-dependent information that would make it unmovable. You can specify:

PO
specifies a partitioned data set organization.

POU
specifies a partitioned data set organization and that the data set contains location-dependent information that makes it unmovable.

If BSAM or QSAM is used to add or retrieve a single member of a partitioned data set, specify DSORG=PS or DSORG=PSU in the BSAM or QSAM DCB. To retrieve a single member of a PDSE, specify DSORG=PS in the BSAM or QSAM DCB. The name of the member being processed in this manner is supplied in the DD statement.

Restrictions are as follows:

- Unmovable data sets cannot be SMS-managed. There are exceptions, however, in cases where the checkpoint/restart function has set the unmovable attribute for data sets that are already

system-managed. This setting prevents data sets that were open when a checkpoint was taken by the application from being moved until you no longer want to perform a restart on that application.

- PDSEs cannot be unmovable data sets.

Source: DSORG parameter must be specified in the DCB macro.

EODAD=*relexp*
specifies the address of the routine given control when the end of the input member is reached. Control is given to this routine when a CHECK macro is issued and the end of the member is reached. If the end of the member is reached but no EODAD address was supplied in the DCB or DCBE, the task is abnormally terminated. The EODAD routine (whether it is specified in the DCBE or DCB) receives control in the addressing mode in which the CHECK macro was issued. For additional information on the EODAD routine, see *z/OS DFSMS Using Data Sets*. This end-of-data routine entry point specified in the DCB must reside below the line. If you wish the entry point to reside above the line, use the EODAD parameter of the DCBE macro. See the EODAD parameter description for the DCBE macro, “DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)” on page 230.

Source: EODAD can be supplied in the DCB macro or by the problem program before the end of the member is reached.

EXLST=*relexp*
specifies the address of the DCB exit list. The EXLST parameter is required if the problem program uses the data control block OPEN exit routine for additional processing or if the DCB ABEND exit is used for abend condition analysis.

The exit list must reside below the line. For the functions, format, and requirements of exit list processing, see *z/OS DFSMS Using Data Sets*. Exit routines can reside above the 16 MB line if you use the technique described in Figure 4 on page 153.

Source: EXLST can be supplied in the DCB macro or by the problem program before the relevant function is needed.

KEYLEN=*absexp* (maximum value is 255)
specifies the length, in bytes, of the key associated with each data block in the direct access storage device data set. If the key length is not supplied from any source by the end of the data control block exit routine, a key length of zero (no keys) is assumed.

A nonzero key length is allowed for input from a PDSE, but is not allowed for output to a PDSE. You can use keys for reading PDSE members, but not for writing PDSE members.

Source: KEYLEN can be supplied in the DCB macro, in the DCB subparameter of a DD statement, by the problem program before the completion of the data control block exit routine, or by the data set label of an existing data set. If KEYLEN=0 is specified in the DCB macro, a special indicator is set in RECFM so that KEYLEN cannot be supplied from the DCB subparameter of a DD statement or data set label of an existing data set. KEYLEN=0 can be coded only in the DCB macro and is ignored if specified in the DD statement.

Key length can be derived from the data class associated with the data set. Key length can also be derived from the JCL keyword LIKE. However, if KEYLEN is specified in the DCB macro, it overrides the value derived from data class or LIKE. For more information, see *z/OS MVS JCL Reference*.

LRECL=*absexp* (maximum value is 32760)
specifies the length, in bytes, for fixed-length records. Or, it specifies the maximum length, in bytes, for variable-length and undefined-length records. The value specified in LRECL cannot exceed the value specified in BLKSIZE.

For PDSEs containing fixed-length blocked records, you must specify LRECL when opened for output. For other types of data sets, you can omit LRECL for BSAM; the system uses the value specified in BLKSIZE. If you want the system to determine the optimum block size for the data set, you must code LRECL. If the LRECL value is coded, it is coded as follows:

Unblocked fixed-length records: the value specified in LRECL must be equal to the value specified in BLKSIZE.

Blocked fixed-length records: the value specified in LRECL must be evenly divisible into the value specified in BLKSIZE. However, except for PDSEs, the LRECL parameter is not checked for validity.

Variable-length records: the value specified in LRECL must include the maximum data length (up to 32752 bytes) plus 4 bytes for the record-descriptor word (RDW).

Undefined-length records: omit LRECL; the actual length is supplied dynamically in a READ/WRITE macro. When an undefined-length record is read, the actual length of the record is returned by the system in the DCBLRECL field of the data control block if your program is not using the large block interface (LBI).

Source: LRECL can be supplied in the DCB macro, in the DCB subparameter of a DD statement, by the problem program before completion of the data control block exit routine, or by the data set label of an existing data set.

Record length can be derived from the data class associated with the data set. Record length can also be derived from the JCL keyword LIKE. For undefined-length records, if LRECL is specified in the DCB macro, it overrides the value derived from data class or LIKE. For more information, see *z/OS MVS JCL Reference*.

MACRF={(R|W|R,W)}
specifies the type of macros (READ, WRITE, and NOTE/POINT) that are used to process the data set. You can specify the following characters for BPAM:

R
specifies that READ macros are to be used. This subparameter automatically allows you to use both the NOTE and POINT macros with the data set. R is required if the OPEN option is INPUT or UPDAT. It has no effect if the OPEN option is OUTPUT or EXTEND.

W
specifies that WRITE macros are to be used. This subparameter automatically allows you to use both the NOTE and POINT macros with the data set. W is required if the OPEN option is OUTPUT or EXTEND. It has no effect if the OPEN option is INPUT. W may be specified if the OPEN option is UPDAT.

All BPAM READ and WRITE macros issued must be tested for completion using a CHECK macro. MACRF does not require any coding to specify that a CHECK macro is to be used.

Source: MACRF must be specified in the DCB macro.

NCP=*absexp* (maximum value is 255)
specifies the maximum number of READ and WRITE macros that are issued before the first CHECK macro is issued to test completion of the I/O operation. In an address space that is constrained for storage below the line, requesting too large a number may result in abnormal termination of the program. If NCP is omitted, 1 is assumed unless you coded the MULTSDN parameter on the DCBE macro.

To request the system to default a value for NCP other than 1, you must supply a DCBE and set MULTSDN to nonzero. The system will update DCBNCP with the system-defaulted NCP (SDN) before the DCB OPEN exit is given control. This allows you to give the system indicators without being dependent on device information such as blocks per track. If you change parameters in the OPEN exit which would cause recalculation of system-determined block size, or you change block size, the SDN will be re-derived after the OPEN exit and stored in the DCBNCP.

Source: NCP can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block open exit routine.

OPTCD={[C][J][W]}
specifies optional services that are performed by the system.

C
specifies that chained scheduling is used. BPAM ignores this obsolete option.

J
specifies that the first data byte in the output data line is a 3800 table reference character. This table reference character selects a particular character arrangement table for the printing of the

data line and can be used singly or with ISO/ANSI or machine control characters. OPEN saves this indication in the data set label and it is available to programs that read the data. For a partitioned data set, the OPTCD value applies to all members.

W
specifies that the system is to perform a validity check for each block written.

OPTCD=W is ignored for PDSEs.

Source: OPTCD can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before an OPEN macro is issued to open the data set. However, all optional services must be requested from the same source.

RECFM={{U[T][A|M]}
{V[B[T]|T][A|M]}
{F[B[T]|T][A|M]}}
specifies the record format and characteristics of the data set being allocated or processed. All the record formats shown above can be specified, but in those record formats that show blocked records, the problem program must perform the blocking and deblocking of logical records. BPAM recognizes only data blocks. You can specify:

A
specifies that the records in the data set contain ISO/ANSI control characters. For a description of control characters, see Appendix C, “Control characters,” on page 397.

B
specifies that the data set contains blocked records.

F
specifies that the data set contains fixed-length records.

M
specifies that the records in the data set contain machine code control characters. For a description of control characters, see Appendix C, “Control characters,” on page 397.

T
specifies that track overflow is used with the data set. Track overflow allows a record to be written partially on one track of a direct access storage device and the remainder of the record to be written on the following track (if required).

Note: This is an obsolete option. The system ignores it.

U
specifies that the data set contains undefined-length records.

V
specifies that the data set contains variable-length records.

Source: RECFM can be supplied in the DCB macro, in the DCB subparameter of a DD statement, by the problem program before completion of the data control block exit routine, or by the data set label of an existing data set.

Record format can be derived from the data class associated with the data set. Record format can also be derived from the JCL keyword LIKE. However, if RECFM is specified in the DCB macro, it overrides the value derived from data class or LIKE. For more information, see *z/OS MVS JCL Reference*.

SYNAD=*relexp*
specifies the address of the error analysis (SYNAD) routine to be given control when an uncorrectable input/output error occurs. The entry point of this SYNAD routine must reside below the line. If you wish the entry point to reside above the line, use the SYNAD parameter of the DCBE macro. You can also use the technique shown in Figure 4 on page 153. The contents of the registers when the error analysis routine is given control are described in *z/OS DFSMS Using Data Sets*. Additional status information available to the SYNAD routine is described in “Status information following an input/output operation” on page 373.

The system detects I/O errors asynchronously. It calls your SYNAD routine synchronously (hence the name SYNAD) when you issue a CHECK macro for the failed block. If SYNAD is omitted in the DCB and DCBE, the task is abnormally terminated when you issue a CHECK and an uncorrectable input/output error occurred.

The error analysis routine must not use the save area pointed to by register 13. The system does not restore registers when it regains control from the error analysis routine. The error analysis routine can issue a RETURN macro that uses the address in register 14 to return control to the system. If control is returned in this manner, the system returns control to the problem program and proceeds as though no error had been found.

SYNAD receives control in the addressing mode in which the CHECK macro was issued. On return from a SYNADAF or SYNADRLS macro issued in the SYNAD routine, the high order byte of register 15 will be unpredictable. Therefore, callers of SYNADAF or SYNADRLS in 31-bit addressing mode must either not use register 15 as a base register or restore the high order bytes on return from SYNADAF or SYNADRLS.

Source: SYNAD can be supplied in the DCB macro or by the problem program. The problem program can also change the error routine address at any time.

DCB—Construct a data control block (BSAM)

The data control block for a basic sequential access method (BSAM) data set is constructed during assembly of the problem program. You must code DSORG and MACRF in the DCB macro, but the other DCB parameters can be supplied to the data control block from other sources. Each DCB parameter description contains a heading, "Source". The information under this heading describes the sources that can supply the parameters. Each reference to a DCB OPEN exit routine also applies to a JFCBE exit routine.

You can assemble the DCB macro into a program that resides above the 16 MB line, but the program must move it below the line before using it. Except for the DCBE, all areas that the DCB refers to, such as EXLST and EODAD, must be below the 16 MB line.

The format of the DCB macro for BSAM is:

[*label*]	DCB	[BFALN={F|D}] [,BFTEK=R] [,BLKSIZE=*absexp*] [,BUFCB=*relexp*] [,BUFL=*absexp*] [,BUFNO=*absexp*] [,BUFOFF={*absexp*|L}] [,DCBE=*relexp*] "1" on page 190 [,DDNAME=*symbol*] "1" on page 190 [,DEVD={{DA [,KEYLEN=*absexp*]} {TA [,DEN={1|2|3|4}] [,TRTCH={C|E|ET|T}|{COMP|NOCOMP}]} {PR [,PRTSP={0|1|2|3}]} {PC [,MODE=[C|E][R]] [,STACK={1|2}] [,FUNC={I|P|PW[XT]|R|RP[D]| RW[T]|RWP[XT][D]|W[T]}] {RD [,MODE=[C|E][O|R]] [,STACK={1|2}] [,FUNC={I|P|PW[XT]|R|RP[D]| RW[T]|RWP[XT][D]|W[T]}]}] ,DSORG={PS|PSU} "1" on page 190 [,EODAD=*relexp*] [,EXLST=*relexp*] [,KEYLEN=*absexp*] [,LRECL={*absexp*|X}] ,MACRF={{(R[C|P])} {(W[C|P|L])} {(R[C|P],W[C|P])}} "1" on page 190 [,NCP=*absexp*] [,OPTCD={{B} {T} {U[C]} {C[T][B][U]} {H[Z][B]} {J[C][U]} {W[C][T][B][U]} {Z[C][T][B][U]} {Q[C][B][T|} {Z}}] [,RECFM={{U[T][A|M]} {V[B][S][T][A|M]} {D[B][S][A]} {F[B|S|T|BS|BT][A|M]}}] [,SYNAD=*relexp*]

Note:

1. This parameter must be supplied before an OPEN macro is issued for this DCB; it cannot be supplied in the open exit routine.

Recommendation: When creating a DCB to open a data set allocated to an SMS-managed volume, do not specify values that would change the data set to a type which cannot be SMS-managed, such as DSORG=PSU.

BSAM supports the following DCB parameters:

BFALN={F|D}
specifies the boundary alignment for each buffer in the buffer pool when the buffer pool is constructed automatically or by a GETPOOL macro. If BFALN is omitted, the system provides doubleword alignment for each buffer.

If the data set being allocated or processed contains ASCII tape records with a block prefix, the block prefix is entered at the beginning of the buffer. Also, data alignment depends on the length of the block prefix. For a description of how to specify the block prefix length, see the description of the DCB BUFOFF that is not L.

You can specify:

F
specifies that each buffer is on a fullword boundary that is not also a doubleword boundary.

D
specifies that each buffer is on a doubleword boundary.

If the BUILD macro is used to construct the buffer pool or if the problem program controls all buffering, the problem program must provide an area for the buffers and control buffer alignment.

Source: BFALN can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. If both BFALN and BFTEK are specified, they must be supplied from the same source.

BFTEK=R
specifies that BSAM is used to read unblocked variable-length spanned records with keys from a direct data set. Each read operation reads one segment of the record and places it in the area designated in the READ macro. The first segment enters at the beginning of the area, but all subsequent segments are offset by the length of the key (only the first segment has a key). The problem program must provide an area in which it can assemble a record, identify each segment, and assemble the segments into a complete record.

Source: BFTEK can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. If both BFTEK and BFALN are specified, they must be supplied from the same source.

BLKSIZE=*absexp* (maximum value KEYLEN + BLKSIZE is 32760)
specifies the maximum block length in bytes. For fixed-length, unblocked records, a non-zero value for this parameter specifies the record length. BLKSIZE includes only the data block length. If keys are used, the length of the key is not included in the value specified for BLKSIZE. If a physical sequential data set that contains fixed-length records (blocked or unblocked) is accessed with a DCB that has specified a DSORG of undefined, then the BLKSIZE value specified must be a value that is less than or equal to the data sets physical block size in the DSCB.

If you wish to process blocks longer than 32760 for BSAM, then use the BLKSIZE keyword on a DCBE macro and use the DCBE keyword on the DCB macro. That requests the large block interface. If the system allows the large block interface, the system modifies the BLKSIZE field in the DCB and your program should not use that field.

The actual value you can specify in BLKSIZE depends on the device type and the record format being used. Device capacity for direct access storage devices is described in Appendix E, “Selecting logical record lengths and block sizes for specific devices,” on page 403. For additional information about device capacity, see the relevant device publication.

When PDSEs, compressed format data sets, or z/OS UNIX files are being processed, the value specified in BLKSIZE can be up to the maximum value. For other data sets on direct access storage devices, the value specified for BLKSIZE cannot exceed the capacity of a single track.

If fixed-length records are used, the value specified in BLKSIZE should be an integral multiple of the value specified for the logical record length (LRECL).

For fixed-length unblocked records, LRECL must equal BLKSIZE (if LRECL is specified).

If variable-length records are used, the value specified in BLKSIZE must include the maximum logical record length (up to 32756 bytes) plus the 4 bytes required for the block descriptor word (BDW). For format-D variable-length records (ASCII data sets), the minimum BLKSIZE value is 18 bytes.

The maximum block size is 32,760 except for Version 3 ISO/ANSI tapes (ISO 1001-1979 and ANSI X3.27-1978), where the maximum block size is 2048. As required by the standard, an attempt to exceed 2048 bytes for a Version 3 tape results in a label validation installation exit being called. The exit may allow violation of the standard by writing larger blocks. This restriction does not apply to Version 4 labels. For more information about the BLKSIZE restrictions, see *z/OS DFSMS Using Data Sets*.

If ASCII tape records with a block prefix are processed, the value specified in BLKSIZE must also include the length of the block prefix.

If BSAM is used to read variable-length spanned records the value specified for BLKSIZE must be as large as the longest possible record segment in the data set, including 4 bytes for the segment descriptor word (SDW) and 4 bytes for the block descriptor word (BDW). The BLKSIZE must equal at least 8 bytes.

If undefined-length records are used, the value specified for BLKSIZE can be altered by the problem program when the actual length becomes known to the problem program. The value can be inserted directly into the DCBBLKSI field of the data control block or specified in the *length* parameter of a READ or WRITE macro.

Processing PDSEs: The system reblocks PDSE records into its own internal format when the data set is written, and reconstructs the blocks using the block size from the DCB when the data set is read. For fixed-length blocked records, the value specified in BLKSIZE *must* be a multiple of the value in LRECL (if LRECL is specified). The LRECL value must be available to OPEN when the PDSE is open for output.

When reading a PDSE directory using fixed-length blocked records, you can specify a BLKSIZE of 256 or greater (the LRECL is ignored). specified in BLKSIZE is the user-perceived block size of the data set. The actual physical (or internal) block size of the data set is calculated by the system when the data set is written. This internal block size is transparent to the user. The system, however, maintains the user's block boundaries when the data is written. Therefore, it is able to reconstruct the exact user blocks when the data set is read. When writing in a compressed format data set, the access method generally compresses the data. This compression and decompression when reading are transparent to the user.

Processing z/OS UNIX files: Block boundaries are not maintained within a z/OS UNIX file. This means that when you read, records may be distributed among blocks differently than they were written. When BLKSIZE is not specified (by any source), it is defaulted to 80 on input.

System-Determined Block Size: IBM recommends that you not specify block size except in these cases:

- Record format is U.
- Medium is tape without standard labels.
- A z/OS UNIX file is being processed.

This makes your program less dependent on the physical characteristics of the device.

System-Determined Block Size for DASD Data Sets: For DASD data sets, if the block size is not specified at the time that the data set is created, and LRECL and RECFM are known, and the record format is not U, the system derives an optimum block size for the data set. This system-determined block size is retained in the data set label. When the data set is opened for output, OPEN checks the block size in the data set label. If it is a system-determined block size, and LRECL or RECFM have

changed from those specified in the data set label, OPEN will re-derive an optimum block size for the data set.

System-Determined Block Size for Tape Data Sets: If you do not specify a block size for a tape data set and the RECFM value is not U, the system determines the optimum block size when the data set is opened for OUTPUT or OUTIN. The system-determined block size depends on the record format and the device type of the tape. See *z/OS DFSMS Using Data Sets* for the table showing the block sizes that are set for tape data sets.

Source: BLKSIZE can be supplied in the DCB or DCBE macro, in the DCB subparameter of a DD statement, by the problem program before completion of the data control block exit routine, by the data set label of an existing data set, or by the system determining a value for a new data set. The system does not copy BLKSIZE when you code the JCL keyword LIKE. It derives the BLKSIZE from RECFM and LRECL which can be copied. For more information on LIKE, see *z/OS MVS JCL Reference* and *z/OS MVS JCL User's Guide*.

BUFCB=*relexp*
specifies the address of the buffer pool control block that you have constructed by issuing a BUILD macro. The buffer pool must reside below the 16MB line.

If the buffer pool is to be constructed automatically or by a GETPOOL macro, omit BUFCB. This is because the system places the address of the buffer pool control block into the data control block. Also, if the problem program is to control all buffering, omit BUFCB. A buffer pool control block resides below the 16MB line.

Source: BUFCB can be supplied in the DCB macro or by the problem program before issuing a GETBUF macro.

BUFL=*absexp* (maximum value is 32760)
specifies the length, in bytes, for each buffer in the buffer pool when the buffer pool is acquired automatically. If BUFL parameter is omitted and you request OPEN to build a buffer pool, the system builds buffers with a length equal to the sum of the values specified in KEYLEN and BLKSIZE. If the problem program requires larger buffers (up to 32760 bytes), BUFL is required. If BUFL is specified, it must be at least as large as the value specified in BLKSIZE in the DCB (without LBI) or in the DCBE (with LBI). If the data set is for card image mode, BUFL should be specified as 160. The description of DEVD contains a description of card image mode.

If the data set contains ASCII tape records with a block prefix, the value specified in BUFL must include the block length plus the length of the block prefix.

If the problem program is to control all buffering or if the buffer pool is to be constructed by a GETPOOL or BUILD macro, BUFL is not required.

Source: BUFL can be supplied in the DCB macro, in the DCB subparameter on a DD statement, or by the problem program before completion of the data control block exit routine.

BUFNO=*absexp* (maximum value is 255)
specifies the number of buffers acquired automatically by the system during OPEN.

If the problem program controls all buffering or if the buffer pool is constructed by a GETPOOL macro, omit BUFNO. The default is 0, meaning the system does not acquire buffers automatically. If the system acquires buffers for BSAM, they reside below the 16MB line. You may obtain each buffer by issuing a GETBUF macro.

Source: BUFNO can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

BUFOFF={*absexp*|L}
specifies the length, in bytes, of the block prefix used with an ASCII tape data set or a tape data set with CCSID. When BSAM is used to read this kind of tape data set, the problem program must use the block prefix length to determine the location of the data in the buffer. When BSAM is used to write an output ASCII tape data set, the problem program must insert the block prefix into the buffer, followed by the data (BSAM considers the block prefix as data). The block prefix without BUFOFF=L and data can consist of any characters that can be converted into 7-bit ASCII code or the CCSID code.

With BUFOFF=L the block prefix is 7-bit ASCII. Any character that cannot be converted is replaced with a substitute character. (For a more detailed description of ASCII conversion characteristics, see *z/OS DFSMS Using Magnetic Tapes*.) For format-D records, the RDW must be binary; if RECFM=D and BUFOFF=L, the RDW and BDW must both be binary. On output, the control program converts the BDW and RDW to ASCII characters and, on input, the control program converts ASCII data to BDW and RDW. This is true even when you use the CCSID parameter to specify a character code other than ASCII. You can specify the following characters in BUFOFF:

absexp
specifies the length, in bytes, of the block prefix. This value can be from 0 to 99 for an input data set. The value must be 0 for writing an output data set with fixed-length or undefined-length records (BSAM considers the block prefix part of the data record).

L
specifies that the block prefix is 4 bytes long and contains the block length. BUFOFF=L is used when format-D records (ASCII) are processed. When BUFOFF=L is specified, the BSAM problem program can process the data records (using READ and WRITE macros) in the same manner as if the data were in format-V variable-length records. For further information on format-D records, see *z/OS DFSMS Using Data Sets*.

If BUFOFF is omitted for an input data set with format-D records, the system inserts the record length into the DCBLRECL field of the data control block. The problem program must obtain the length from this field to process the record.

If BUFOFF is omitted from an output data set with format-D records, the problem program must insert the actual record length into the DCBBLKSI field of the data control block or specify the record length in the *length* parameter of a WRITE macro.

Source: BUFOFF can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before an OPEN macro is issued to open the data set. BUFOFF=*absexp* can also be supplied by the label of an existing data set. BUFOFF=L cannot be supplied by the label of an existing data set.

DCBE=*relexp*
specifies the address of a DCB Extension (DCBE). The DCBE may reside above the 16MB line. You may assemble a DCB and DCBE in a program that resides above the line if the DCB is copied below the line before opening the copy.

If the DCBE is specified, it must be specified before issuing the OPEN macro. Like the DCB, the DCBE must exist until the data set is closed. Otherwise, there may be unpredictable results.

Only one open DCB at a time can refer to a particular DCBE. After a DCB is successfully closed, you can open a different DCB referring to the DCBE.

The DCBE is not required with BSAM unless the data set requires a DCBE option or if you choose to use DCBE options.

If a DCBE exists, the flags DCBH0 and DCBH1 are both set on. The pointer to the DCBE is stored at offset +0 in the DCB (and replaces the field DCBRELAD). If a DCBE exists, data that would be stored at DCBRELAD is stored in the DCBE (DCBERELA). If a DCBE does not exist, DCBRELAD continues to be located at offset +0 in the DCB.

Source: You can supply the DCBE address in the DCB macro or before issuing an OPEN macro to open the data set.

DDNAME=*symbol*
specifies the name that is to be used to identify the job control language data definition (DD) statement that defines the data set being allocated or processed.

Source: DDNAME can be supplied in the DCB macro or by the problem program before an OPEN macro is issued to open the data set.

DEVD={DA|TA|PR|PC|RD}
specifies the device type where the data set can or does reside. The device types above are shown with the optional parameters that can be coded when a particular device is used. The devices are

listed in order of device independence. For example, if you code DEVD=DA in a DCB macro (or omit DEVD parameter, which causes a default to DA), you can later use the data control block constructed during assembly for any of the other devices, but, if you code DEVD=RD, you can use the data control block only with a card reader or card reader punch. Unless you are certain that device interchangeability is not required, you should either code DEVD=DA or omit the parameter and allow it to default to DA.

Restriction: If the data set can or does reside on DASD, do not code a value other than DEVD=DA. For spooled data sets, dummy data sets, and TSO terminals any DEVD value is acceptable.

DEVD is discussed below according to individual device type:

DEVD=DA
[,KEYLEN=*absexp*]
specifies that the data control block can be used for a direct access storage device (or any of the other device types described following DA).

KEYLEN=*absexp*
can be specified only for data sets that reside on direct access storage devices. Because the KEYLEN is usually coded without the DEVD parameter (default taken), the description of KEYLEN is in alphabetic sequence with the other parameters.

DEVD=TA
[,DEN={1|2|3|4}]
[,TRTCH={C|E|ET|T}|{COMP|NOCOMP}]
specifies that the data control block can be used for a magnetic tape data set (or any of the other device types described following TA). If TA is coded, you can code the following optional parameters:

DEN={1|2|3|4}
specifies the recording density in the number of bits-per-inch per track as follows:

DEN	7-Track	9-Track	18-Track	36-Track
1	556	N/A	N/A	N/A
2	800	800 (NRZI)"1" on page 195	N/A	N/A
3	N/A	1600 (PE)"2" on page 195	N/A	N/A
4	N/A	6250 (GCR)"3" on page 195	N/A	N/A

Notes:

1. NRZI is for nonreturn-to-zero inverted mode.
2. PE is for phase encoded mode.
3. GCR is for group coded recording mode.

If DEN is not supplied by any source, the highest applicable density is assumed.

For magnetic tape drives that use cartridges, such as the 3480, only a single density is available and is used by the system for reading and writing; any density with the DEN parameter is ignored.

TRTCH={C|E|ET|T}|{COMP|NOCOMP}
The TRTCH parameter has two different sets of values. One of the sets, {C|E|ET|T}, is used to specify the recording technique for 7-track tape. The other set, {COMP|NOCOMP}, is used to specify the recording technique for magnetic tape drives with Improved Data Recording Capability and override the system default.

{C|E|ET|T}
These values specify the recording technique for 7-track tape. One of the above four values can be coded. If TRTCH is omitted, odd parity with no translation or conversion is assumed. You can specify:

C
specifies that the data-conversion feature is used with odd parity and no translation.

E
specifies even parity with no translation or conversion.

ET
specifies that even parity with BCDIC to EBCDIC translation is required and no data-conversion feature.

T
specifies that BCDIC to EBCDIC translation is required with odd parity and no data-conversion feature.

{COMP|NOCOMP}
These values specify the recording technique for magnetic tape drives with Improved Data Recording Capability. Either of the two values can be coded. If TRTCH is omitted, the system default specified in the active DEVSUPyy member of SYS1.PARMLIB (initially set to NOCOMP) is assumed. You can specify:

COMP
record data in compacted format. COMP is not supported with ISO/ANSI tape labels.

NOCOMP
record data in standard format.

Source: TRTCH can be supplied in the DCB macro, in the DCB subparameter on a DD statement, in the IBM standard tape label or by the problem program before completion of the data control block exit routine.

DEVD=PR
[,PRTSP={0|1|2|3}]
specifies that the data control block is used for a directly allocated printer (or any of the other device types following PR). This has no effect for a spooled (SYSOUT) or subsystem data set. If PR is coded, you can specify:

PRTSP={0|1|2|3}
specifies the line spacing on the printer. This parameter is not valid if the RECFM parameter specifies either machine (RECFM=M) or ISO/ANSI (RECFM=A) control characters. If PRTSP is not specified from any source, 1 is assumed. You can specify:

0
specifies that spacing is suppressed (no space).

1
specifies single spacing.

2
specifies double spacing (one blank line between printed lines).

3
specifies triple spacing (two blank lines between printed lines).

DEVD=PC
[,MODE=[C|E][R]]
[,STACK={1|2}]
[,FUNC={I|P|PW[XT]|R|RP[D]|RW[T] |RWP[XT][D]| W[T]}]
specifies that the data control block is used for a card punch (or any of the other device types following PC). If PC is coded, you can specify the following optional parameters:

MODE=[C|E][R]
specifies the mode of operation for the card punch. You can specify the following characters (if MODE is omitted, E is assumed):

C
specifies that the cards are punched in column binary (card image) mode. In column binary mode, the 12 rows in each card column are punched from 2 consecutive bytes in virtual storage. Rows 12 through 3 are punched from the low-order 6 bits of one byte and rows 4 through 9 are punched from the low-order 6 bits of the following byte.

E
specifies that the cards are punched in EBCDIC code.

R
specifies that the program runs in read-column-eliminate mode (3525 Card Punch, read feature).

If you code R for the MODE subparameter of the DCB parameter of the DD statement, you must also code either C or E.

STACK={1|2}
specifies that the stacker bin where the card is placed after punching is completed. If this parameter is omitted, stacker number 1 is used. You can specify:

1
specifies stacker number 1.

2
specifies stacker number 2.

FUNC={I|P|PW[XT]|R|RP[D]| RW[T]|RWP[XT][D]|W[T]}
defines the type of 3525 card punch data sets used. If the FUNC parameter is omitted from all sources, a data set opened for input defaults to read only, and a data set opened for output defaults to punch only. You can specify:

D
specifies that the data protection option is used. The data protection option prevents punching information into card columns that already contain data. When the data protection option is used, an 80-byte data protection image (DPI) must have been previously stored in SYS1.IMAGELIB. Specify its name in the FCB parameter of the DD statement. Data protection applies only to the output/punch portion of a read and punch or read, punch, and print operation.

I
specifies that the data in the data set is punched into and printed on the cards. The first 64 characters are printed on line 1 of the card and the remaining 16 characters are printed on line 3.

P
specifies that the data set is for punching cards. See the description of the character X for associated punch and print data sets.

R
specifies that the data set is for reading cards.

T
specifies that the two-line print option is used. The two-line print option allows two lines of data to be printed on the card (lines 1 and 3). If T is not specified, the multiline print option is used; this allows printing on all 25 possible print lines. In either case, the data printed can be the same as the data punched in the card, or it can be entirely different data.

W
specifies that the data set is for printing. See the description of the character X for associated punch and print data sets.

X
specifies that an associated data set is opened for output for both punching and printing. Coding the character X is used to distinguish the 3525 printer output data set from the 3525 punch output data set.

DEVD=RD
[,MODE=[C|E][O |R]]
[,STACK={1|2}]
[,FUNC={I|P|PW[XT]|R|RP[D]| RW[T]|RWP[XT][D]| W[T]}]
specifies that the data control block is used with a card reader or card read punch. If RD is specified, the data control block cannot be used with any other device type. When RD is coded, you can specify the following optional parameters:

MODE=[C|E][O|R]
specifies the mode of operation for the card reader. You can specify:

C
specifies that the cards to read are in column binary (card image) mode. In column binary mode, the 12 rows in each card column are read into 2 consecutive bytes of virtual storage. Rows 12 through 3 are read into one byte and rows 4 through 9 are read into the following byte.

E
specifies that the cards to read contain data in EBCDIC code.

O
specifies that the program runs in optical-mark-read mode (3505 Card Reader).

R
specifies the program runs in read-column-eliminate mode (3505 Card Reader or 3525 Card Punch, read feature).

If you code R or 0 for the MODE subparameter of the DCB parameter of the DCB parameter of the DD statement, you must also code either C or E.

STACK={1|2}
specifies the stacker bin where the card is placed after reading is completed. If this parameter is omitted, stacker number 1 is used. You can specify:

1
specifies stacker number 1.

2
specifies stacker number 2.

FUNC={I|P|PW[XT]|R|RP[D]| RW[T]|RWP[XT][D]|W[T]}
defines the type of 3525 card punch data sets used. If the FUNC parameter is omitted from all sources, a data set opened for input defaults to read only, and a data set opened for output defaults to punch only. You can specify:

D
specifies that the data protection option is used. The data protection option prevents punching information into card columns that already contain data. When the data protection option is used, an 80-byte data protection image (DPI) must have been previously stored in SYS1.IMAGELIB. Specify its name in the FCB parameter of the DD statement. Data protection applies only to the output/punch portion of a read and punch or read, punch, and print operation.

I
specifies that the data in the data set is punched into and printed on the cards. The first 64 characters are printed on line 1 of the card and the remaining 16 characters are printed on line 3.

P
specifies the data set is for punching cards. See the description of the character X for associated punch and print data sets.

R
specifies that the data set is for reading cards.

T
specifies that the two-line print option is used. The two-line print option allows two lines of data to be printed on the card (lines 1 and 3). If T is not specified, the multiline print option is used; this allows printing on all 25 possible print lines. In either case, the data printed can be the same as the data punched in the card, or it can be entirely different data.

W
specifies that the data set is for printing. See the description of the character X for associated punch and print data sets.

X
specifies that an associated data set is opened for output for both punching and printing. Coding the character X is used to distinguish the 3525 printer output data set from the 3525 punch output data set.

Source: DEVD can be supplied only in the DCB macro. However, the optional parameters can be supplied in the DCB macro, the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

DSORG={PS|PSU}
specifies the data set organization and whether the data set contains any location-dependent information that would make it unmovable. You can specify:

PS
specifies a physical sequential data set.

PSU
specifies a physical sequential data set containing location-dependent information that makes it unmovable. See "NOTE—Provide relative position (BPAM and BSAM—tape and DASD only)" on page 284 for more information about unmovable data sets.

Restriction: Unmovable data sets cannot be system-managed. There are exceptions, however, in cases where the checkpoint/restart function has set the unmovable attribute for data sets that are already system-managed. This setting prevents data sets opened previously by a checkpointed application from being moved until you no longer want to perform a restart on that application. PDSEs and extended format data sets must be system-managed, and, thus, cannot be unmovable.

Source: You must code DSORG in the DCB macro.

EODAD=*relexp*
specifies the address of the routine given control when the end of an input data set is reached. If the record format is RECFM=FS or FBS, the end-of-data condition is sensed when a file mark is read or when more data is requested after reading a truncated block. The end-of-data routine is entered when the CHECK macro determines that the READ macro reached the end of the data. If the end of the data set is reached but no EODAD address was supplied to the data control block (DCB) or DCBE, the task is abnormally terminated. For additional information on the EODAD user exit routine, see *z/OS DFSMS Using Data Sets*.

When the data set has been opened for other than UPDAT, the system automatically switches volumes when the end of data on each volume is reached.

When the data set has been opened for UPDAT and volumes are to be switched, the problem program should issue a FEOV macro after the EODAD routine has been entered.

This end-of-data routine entry point specified in the DCB must reside below the line. If you wish the entry point to reside above the line, use the EODAD parameter of the DCBE macro. See the EODAD parameter description for the DCBE macro, "DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)" on page 230. The EODAD routine (whether it is specified in the DCBE or DCB) receives control in the addressing mode in which the CHECK macro was issued.

Source: EODAD can be supplied in the DCB macro or by the problem program before the end of the data set is reached.

EXLST=*relexp*
specifies the address of the DCB exit list. EXLST is required if the problem program requires additional processing for user labels, user totaling, data control block OPEN exit routines, end-of-volume, block count exits, defining a forms control buffer (FCB) image, using the JFCBE exit (for the IBM 3800 Printing Subsystem), or using the DCB ABEND exit for abend condition analysis.

The exit list must reside below the line. For the function, format, and requirements of exit list processing, see *z/OS DFSMS Using Data Sets*. Exit routines can reside above the 16 MB line if you use the technique described in Figure 4 on page 153.

Source: EXLST can be supplied in the DCB macro or by the problem program any time before the relevant function is needed.

KEYLEN=*absexp* (maximum value is 255)
specifies the length, in bytes, for the key associated with each data block in a direct access storage device data set. If the key length is not supplied from any source before completion of the data control block exit routine, a key length of zero (no keys) is assumed.

A nonzero key length is allowed for input from a PDSE, but is not allowed for output to a PDSE. You can use keys for reading PDSE members, but not for writing PDSE members.

You cannot specify a nonzero key length on output for an extended format data set. KEYLEN is ignored for z/OS UNIX files.

Source: KEYLEN can be supplied in the DCB macro, in the DCB subparameter of a DD statement, by the problem program before the completion of the data control block exit routine, or by the data set label of an existing data set. If KEYLEN=0 is specified in the DCB macro, a special indicator is set in RECFM so that KEYLEN cannot be supplied from the DCB subparameter of a DD statement or data set label of an existing data set. KEYLEN=0 can be coded only in the DCB macro and is ignored if specified in the DD statement.

Key length can be derived from the data class associated with the data set. Key length can also be derived from the JCL keyword LIKE. However, if KEYLEN is specified in the DCB macro, it overrides the value derived from data class or LIKE. For more information, see *z/OS MVS JCL Reference*.

LRECL={*absexp*|X}
specifies the length, in bytes, for fixed-length records, or it specifies the maximum length, in bytes, for variable-length records. LRECL=X is used for variable-length spanned records that exceed 32756 bytes. Except when variable-length spanned records are used, the value specified in LRECL cannot exceed the value specified in BLKSIZE.

LRECL is required when using variable-length spanned records. LRECL is also required for PDSEs and compressed format data sets containing fixed-length block records when opened for output.

For other types of data sets. LRECL can be omitted for BSAM; the system uses the value specified in BLKSIZE. If you want the system to determine the optimum block size for the data set, you must code LRECL. If an LRECL value is coded, it is coded as follows:

Unblocked fixed-length records: the value specified in LRECL must be equal to the value specified in BLKSIZE.

Blocked fixed-length records: the value specified in the LRECL parameter must be evenly divisible into the value specified in the BLKSIZE parameter. However, except for PDSEs and compressed format data sets, LRECL is not checked for validity.

Variable-length records: the value specified in LRECL must include the maximum data length (up to 32752 bytes) plus 4 bytes for the record-descriptor word (RDW).

Undefined-length records: omit LRECL; the actual length is supplied dynamically in a READ/WRITE macro. When an undefined-length record is read, the actual length of the record is returned by the system in the DCBLRECL field of the data control block if your program is not using the large block interface (LBI).

z/OS UNIX files: record boundaries are not maintained within a binary z/OS UNIX file. When LRECL is not specified (by any source), it is defaulted to 80 on input.

X
When using BSAM to create a direct data set with variable-length spanned records, the LRECL value should be the maximum data length (up to 32752) plus 4 bytes for the record descriptor word (RDW). Specify LRECL=X if the logical record length is greater than 32756 bytes.

Source: LRECL can be supplied in the DCB macro, in the DCB subparameter of a DD statement, by the problem program before completion of the data control block exit routine, or by the data set label of an existing data set.

Record length can be derived from the data class associated with the data set. Record length can also be derived from the JCL keyword LIKE. However, if LRECL is specified in the DCB macro, it overrides the value derived from data class or LIKE. For more information, see *z/OS MVS JCL Reference*.

MACRF={{(R[C|P])}
{(W[C|P|L])}
{(R[C|P],W[C|P])}}
specifies the type of macros (READ, WRITE, CNTRL, and NOTE/POINT) that are used with the data set being created or processed. The BSAM MACRF parameter also provides the special form (MACRF=WL) for creating a direct data set. MACRF can be coded in any of the combinations shown above. The following characters can be coded for BSAM:

C
specifies that the CNTRL macro is used with the data set. If you specify C, the device must be one of these described in "CNTRL—Control directly allocated input/output device (BSAM and QSAM)" on page 170. If C is specified for use with a card reader, a CNTRL macro must follow each input request.

L
specifies that BSAM is used to create a direct data set. This character can be specified only in the combination MACRF=WL. This does not support 31-bit addressing.

P
specifies that POINT macros are used with the data set being created or processed. Specifying P in MACRF also automatically allows you to use NOTE macros with the data set.

Do not code P for FIFO or character special z/OS UNIX files or with PATHOPTS=OAPPEND (see NOTE and POINT macros for more information).

The NOTE and POINT macros cannot be used with spooled data sets. Some subsystems (SUBSYS on the DD statement) may support the NOTE and POINT macros with TYPE=REL specified or defaulted. Assume it does not work unless the subsystem documentation says it is supported.

R
specifies that READ macros are used. R is required if the OPEN option is INPUT, UPDAT, or RDBACK. It has no effect if the OPEN option is OUTPUT or EXTEND. R may be specified if the OPEN option is INOUT or OUTIN.

W
specifies that WRITE macros are used. W is required if the OPEN option is OUTPUT or EXTEND. It has no effect if the OPEN option is INPUT or RDBACK. W may be specified if the OPEN option is UPDAT, INOUT, or OUTIN.

Rule: You must use the CHECK macro to check each READ and WRITE macro issued in the problem program for completion.

Source: MACRF must be specified in the DCB macro.

NCP=*absexp* (maximum value is 255)
specifies the maximum number of READ and WRITE macros that are issued before the first CHECK macro is issued to test for completion of the I/O operation. In an address space that is constrained

for storage below the line, requesting too large a number may result in abnormal termination of the program. If NCP is omitted, 1 is assumed unless you coded the MULTSDN parameter on the DCBE macro.

To request the system to default a value for NCP other than 1, you must supply a DCBE and set MULTSDN to nonzero. The system will update DCBNCP with the system-defaulted NCP (SDN) before the DCB OPEN exit is given control. This allows you to give the system indicators without being dependent on device information such as blocks per track or number of stripes. If you change parameters in the OPEN exit which would cause recalculation of system-determined block size, or you change block size, the SDN will be re-derived after the OPEN exit and stored in the DCBNCP.

If NCP is zero and MULTSDN is non-zero and the block size is greater than or equal to 32768, then NCP will be set to the MULTSDN value with a minimum of 2 and a maximum of 16.

Extended format data sets: Some programs calculate NCP in the DCB OPEN exit by using TRKCALC to get the number of blocks per track. Since a suffix is included with each block on DASD for an extended format data set, the number of blocks per track returned by TRKCALC might not be accurate because it does not take into account the block suffix. This may result in allocating more buffers than is necessary for an extended format data set which consists of only one stripe. Also, for extended format data sets which consist of more than one stripe, using an NCP of this number of blocks per track will result in inadequate performance unless NCP is made larger based on the number of stripes.

Recommendation: For compressed format data sets, do not specify NCP (thus, allowing the system to default it to 1) or specify NCP=1. This is the optimal value for NCP for a compressed format data set since the system handles all buffering internally for these data sets. Therefore, the following technique for choosing a value for NCP does not pertain to compressed format data sets. In fact, since the physical blocks have no relationship to the user-specified block size, IBM recommends that TRKCALC not be used to return number of blocks per track of a compressed format data set, since the value returned will not be accurate.

If you choose to calculate NCP in the DCB OPEN exit, then you may want to choose to use the following technique to calculate a value for extended format data sets. However, you can gain the same effect by coding the MULTSDN parameter on the DCBE macro.

- Code a DCBE.
- In the OPEN exit, determine if the data set is extended format (the value in DCBENSTR will be nonzero if the data set is extended format). If the data set is not extended format, then OPEN will set DCBENSTR to 0.
- Issue a DEVTYPE macro with the INFO=SUFFIX parameter to obtain the length of the suffix.
- Add DCBBLKSI and the length of the suffix and pass this number in to TRKCALC to get the correct number of blocks per track.
- Multiply the number of blocks per track from TRKCALC by the number of stripes of an extended format data set (DCBENSTR). Assuming this number of buffers is used, this would give one track's worth of buffers per stripe.

  In addition, you may choose to multiply this value by n to get an NCP value which is n tracks worth of buffers per stripe. A value of n greater than 1 is likely to improve performance.
- If the calculated value exceeds 255, decrease it appropriately. Store the calculated NCP value in DCBNCP.

Source: NCP can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block open exit routine.

OPTCD={{B}
{T}
{U[C]}
{C[T][B][U]}
{H[Z][B]}
{J[C][U]}
{W[C][T][B][U]}
{Z[C][T][B][U]}
{Q[C][B][T}

specifies the optional services that are used with the sequential data set. Two of the optional services, OPTCD=B and OPTCD=H, cannot be specified in the DCB macro. They are requested in the DCB subparameter of a DD statement. Because all optional services requests must be supplied by the same source, you must omit OPTCD from the DCB macro if either of these options is requested in a DD statement.

Note: If you specify OPTCD=B on the DD statement for a multivolume tape data set, the system will generate the equivalent of individual concatenated DD statements for each volume serial number. This means that the system allocates one tape drive for each volume.

You can code the following characters in any order, in any combination, and without commas between characters.

C

specifies that chained scheduling is used. OPTCD=C cannot be specified if BFTEK=R is specified for the same data control block. Also, chained scheduling cannot be specified for associated data sets or printing on a 3525 and is ignored for direct access storage devices.

Note: Except where it is not allowed, chained scheduling is used whether requested or not. For conditions under which chained scheduling is not allowed, see *z/OS DFSMS Using Data Sets*.

J

specifies that the first data byte in the output data line is a 3800 table reference character. This table reference character selects a particular character arrangement table for the printing of the data line and can be used singly or with ISO/ANSI or machine control characters. This option has effect for DASD data sets, SYSOUT data sets, and a directly allocated IBM 3800 Printing Subsystem or IBM 3900 Printing Subsystem. On DASD, this indication is saved in the data set label and can be available to programs that read the data. For a partitioned data set, the OPTCD value applies to all members. If the SYSOUT data set is printed on a device that does not support table reference character, the system discards that byte.

Q

requests conversion of the tape records between what is stored on tape and what is supplied from/to the problem program. For input requests, conversion is done at CHECK time. For output requests, conversion is done just before the record is written to tape. For further information on this conversion, see *z/OS DFSMS Using Data Sets*.

The Q option implies that the character representation of the data on tape differs from that seen by the problem program. Data management converts records according to one of the following techniques:

- **CCSID Conversion**

  If CCSIDs are supplied from any source for ISO/ANSI V4 tapes, records are converted between the CCSID which represents the data on tape and the CCSID as seen by the problem program. You can also prevent conversion by supplying a special CCSID.

- **Default Character Conversion**

  If you are using non-ISO/ANSI V4 tapes or if CCSIDs are not supplied by any source, data management converts the records between ASCII code (which represents the data on tape) to EBCDIC code (which is seen by the problem program) using specific tables defined for this default character conversion.

Refer to *z/OS DFSMS Using Data Sets*, for a complete description of CCSID conversion and default character conversion.

See *z/OS DFSMS Using Magnetic Tapes* for more information about ISO/ANSI labels.

Q is supported only for a magnetic tape that does not have IBM standard labels. If the tape has ISO/ANSI labels (LABEL=(,AL)), the system assumes OPTCD=Q.

T

specifies the user totaling function. If this function is requested, EXLST should specify the address of an exit list that includes a user totaling entry. T cannot be specified for SYSIN and SYSOUT data sets.

Restriction: User totaling is supported only for sequential data sets that are not extended format data sets. If you specify user totaling for a partitioned data set, a PDSE, an extended format data set, or a z/OS UNIX file, it is ignored.

U

is specified only for a printer with the universal character set (UCS) feature or the 3800 Printing Subsystem. This option unblocks data checks (permits them to be recognized as errors) and allows analysis by the appropriate error analysis routine (SYNAD exit routine). If the U option is omitted, data checks are not recognized as errors.

W

specifies, for DASD, that the system is to perform a validity check on each block written on a direct access storage device.

OPTCD=W is ignored for PDSEs, extended format data sets, and z/OS UNIX files.

The system reads each block back. The intent is to ensure that the data would survive a subsequent power failure. Because of the performance degradation and the reliability of modern IBM devices and recovery techniques, IBM recommends not coding OPTCD=W.

For buffered tape devices, specifies that device end interrupt is given only when a block is physically on the device. By specifying OPTCD=W with buffered devices, you do not benefit from the performance advantage of buffering.

Z

for magnetic tape (input only). Requests the system to shorten its normal error recovery procedure to consider a data check as a permanent I/O error after five unsuccessful attempts to read a record. OPTCD=Z is intended for use when a tape is known to contain errors and there is no need to process every record. The error analysis routine (SYNAD) should keep a count of permanent errors and terminate processing if the number becomes excessive.

Note: The following optional services can be requested in the DCB subparameter of a DD statement. If either of these options is requested, the complete OPTCD parameter must be supplied in the DD statement.

B

forces the end-of-volume (EOV) routine to disregard the end-of-file recognition for magnetic tape. When this occurs, the EOV routine uses the number of volume serial numbers to determine end of file. For an input data set on a standard labeled (SL or AL) tape, the EOV routine treats EOF labels as EOV labels until the volume serial list is exhausted. After all the volumes have been read, control is passed to your end-of-data routine. This option allows SL or AL tapes to be read out of volume sequence or to be concatenated to another tape using one DD statement.

H

specifies that the VSE/MVS interchange feature is being used with the data set. It is on magnetic tape and may contain VSE embedded checkpoint records. You cannot use this option with LBI.

Source: OPTCD can be supplied in the DCB macro, in the DCB subparameter of a DD statement, in the data set label for direct access storage devices, or by the problem program before completion

of the DCB open exit routine or JFCBE exit routine. However, all optional services must be requested from the same source.

RECFM={{U[T][A|M]}
{V[B][S][T][A|M]}
{D[B][S][A]}
{F[B|S|T|BS|BT][A|M]}}

specifies the record format and characteristics of the data set being allocated or processed. All the record formats shown above can be specified. BSAM recognizes only data blocks. Therefore, for record formats that specify blocked records, the problem program must block and deblock logical records. You can specify:

A
specifies that the records in the data set contain International Organization for Standardization (ISO) or American National Standards Institute (ANSI) control characters. For a description of control characters, see Appendix C, “Control characters,” on page 397.

B
specifies that the data set contains blocked records.

D
specifies that the data set contains variable-length ASCII tape records.

F
specifies that the data set contains fixed-length records.

M
specifies that the records in the data set contain machine code control characters. For a description of control characters, see Appendix C, “Control characters,” on page 397. RECFM=M cannot be used with ASCII data sets.

S
specifies, for fixed-length records, that the records are written as standard blocks. Except for the last block or track in the data set, the data set contains no truncated blocks or unfilled tracks. Do not code S to retrieve fixed-length records from a data set allocated using a RECFM other than standard.

For variable-length records, including variable-length ASCII, S specifies that a record can span more than one block.

T
specifies that track overflow is used with the data set. Track overflow allows a record to be written partially on one track of a direct access storage device and the remainder of the record to be written on the following tracks (if required).

Note: This is an obsolete option. The system ignores it.

U
specifies that the data set contains undefined-length records.

Restriction: Format-U records are not supported for Version 3 or Version 4 ISO/ANSI tapes. An attempt to process a format-U record for a Version 3 or Version 4 tape results in a label validation installation exit being called.

On ISO/ANSI Version 1 (ISO 1001-1969 or ANSI X3.27-1969) tapes, format-U records can be used for input only. These records are the same as in other types of data sets except that any control characters must be ISO/ANSI control characters and block prefixes can be used.

V
specifies that the data set contains variable-length records.

Restrictions are as follows:

- Do not specify RECFM=FS or RECFM-FBS for a partitioned data set or PDSE because it will cause an abend.
- RECFM=V cannot be specified for a card reader data set or an ISO/ANSI tape data set.

- RECFM=VS, VBS, DS, or DBS do not provide the spanned record function. If this format is used, the problem program must block and segment the records.
- RECFM=VS, VBS, DS, or DBS cannot be specified for a SYSIN data set.
- RECFM=VS or VBS cannot be specified for a z/OS UNIX file.
- RECFM=V cannot be used for a 7-track tape unless the data conversion feature (TRTCH=C) is used.

Source: RECFM can be supplied in the DCB macro, in the DCB subparameter of a DD statement, by the problem program before completion of the data control block exit routine, or by the data set label of an existing data set.

Record format can be derived from the data class associated with the data set. Record format can also be derived from the JCL keyword LIKE. However, if RECFM is specified in the DCB macro, it overrides the value derived from data class or LIKE. For more information, see *z/OS MVS JCL Reference*.

SYNAD=*relexp*
specifies the address of the error analysis (SYNAD) routine to be given control when an uncorrectable input/output error occurs. The entry point of this SYNAD routine must reside below the line. If you wish the entry point to reside above the line, use the SYNAD parameter of the DCBE macro. You can also use the technique shown in Figure 4 on page 153. The contents of the registers when the error analysis routine is given control are described in *z/OS DFSMS Using Data Sets*. Additional status information available to the SYNAD routine is described in "Status information following an input/output operation" on page 373.

The system detects I/O errors asynchronously. It calls your SYNAD routine synchronously (hence the name SYNAD) when you issue a CHECK macro for the failed block. If SYNAD is omitted in the DCB and DCBE, the task is abnormally terminated when you issue a CHECK and an uncorrectable input/output error occurred.

The error analysis routine must not use the save area pointed to by register 13. The system does not restore registers when it regains control from the error analysis routine. The error analysis routine can issue a RETURN macro that uses the address in register 14 to return control to the system. If control is returned in this manner, the system returns control to the problem program and proceeds as though no error had been found.

The SYNAD routine (whether it is specified in the DCBE or DCB) receives control in the addressing mode in which the CHECK macro was issued. On return from a SYNADAF or SYNADRLS macro issued in the SYNAD routine, the high order byte of register 15 will be unpredictable. Therefore, callers of SYNADAF or SYNADRLS in 31-bit addressing mode must either not use register 15 as a base register or restore the high order bytes on return from SYNADAF or SYNADRLS.

When operating a directly allocated IBM 3800 Model 3, 6, or 8 using all-points addressability, the SYNAD exit routine is entered if Print Services Facility (PSF) detects an unrecoverable error. However, no error information is available to the SYNAD routine. If you want to continue processing, you must close and reopen the data set to restart PSF.

Source: SYNAD can be supplied in the DCB macro or by the problem program. The problem program can also change the error routine address at any time.

DCB—Construct a data control block (QISAM interface to VSAM)

The data control block for a queued indexed sequential access method (QISAM interface to VSAM) data set is constructed during assembly of the problem program. You must code DSORG and MACRF in the DCB macro, but the other DCB parameters can be supplied to the data control block from other sources. Each QISAM DCB parameter description contains a heading, "Source". The information under this heading describes the sources that can supply the parameter. Each reference to a DCB OPEN exit routine applies also to a JFCBE exit routine.

Recommendation: The system no longer supports indexed sequential data sets. Convert the data set to a key sequenced data set (KSDS) and use the ISAM interface of VSAM or convert your program to use VSAM.

You can assemble the DCB macro into a program that resides above the 16 MB line, but the program must move it below the line before using it.

The format of the DCB macro for QISAM is:

[*label*]	DCB	[BFALN={F|D}] [,BLKSIZE=*absexp*] [,BUFCB=*relexp*] [,BUFL=*absexp*] [,BUFNO=*absexp*] [,CYLOFL=*absexp*] [,DDNAME=*symbol*] "1" on page 207 ,DSORG={IS|ISU} [,EODAD=*relexp*] [,EXLST=*relexp*] [,KEYLEN=*absexp*] [,LRECL=*absexp*] ,MACRF={{(PM)} {(PL)} {(GM[,S{K|I}])} {(GL[,S{K|I}][,PU])}} [,NTM=*absexp*] [,OPTCD={[I][L][M][R][U][W][Y]}] [,RECFM={V[B]|F[B]}] [,RKP=*absexp*] [,SYNAD=*relexp*]

Note:

1. This parameter must be supplied before an OPEN macro is issued for this DCB; it cannot be supplied in the open exit routine.

QISAM supports the following DCB parameters:

BFALN={F|D}
specifies the boundary alignment of each buffer in the buffer pool when the buffer pool is constructed automatically or by a GETPOOL macro. If BFALN is omitted, the system provides doubleword alignment for each buffer. You can specify:

F
specifies that each buffer is on a fullword boundary that is not also a doubleword boundary.

D
specifies that each buffer is on a doubleword boundary.

If the BUILD macro is used to construct the buffer pool, the problem program must provide a storage area for the buffers and control buffer alignment.

Source: BFALN can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

BLKSIZE=*absexp* (maximum value KEYLEN + BLKSIZE is 32760)
specifies the length, in bytes, for each data block when fixed-length records are used. Or, it specifies the maximum length in bytes, for each data block when variable-length records are used. You must specify the BLKSIZE parameter when creating an indexed sequential data set. When processing an existing indexed sequential data set, you must omit BLKSIZE (it is supplied by the data set label).

If fixed-length records are used, the value specified in BLKSIZE must be a whole number multiple of the value specified in LRECL.

Source: When an indexed sequential data set is allocated, the BLKSIZE can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. The system does not copy BLKSIZE when you code the JCL keyword LIKE. It derives the BLKSIZE from RECFM and LRECL which can be copied. When an existing indexed sequential data set is processed, BLKSIZE must be omitted from the other sources, allowing the data set label to supply the value.

BUFCB=*relexp*
specifies the address of the buffer pool control block that constructed by a BUILD macro.

If the system builds the buffer pool automatically or if the buffer pool is built by a GETPOOL macro, omit BUFCB, because the system places the address of the buffer pool control block into the data control block.

Source: BUFCB can be supplied in the DCB macro or by the problem program before completion of the data control block exit routine.

BUFL=*absexp* (maximum value is 32760)
specifies the length, in bytes, of each buffer in the buffer pool to be constructed by a BUILD or GETPOOL macro. When the data set is opened, the system computes the minimum buffer length required and verifies that the length in the buffer pool control block is equal to or greater than the minimum length required. The system then inserts the computed length into the data control block.

BUFL is not required for QISAM if the system acquires buffers automatically, because the system computes the minimum buffer length required and inserts the value into the data control block.

If the buffer pool is constructed with a BUILD or GETPOOL macro, additional space is required in each buffer for system use. For a description of the buffer length required for various ISAM operations, see *z/OS DFSMS Using Data Sets*.

Source: BUFL can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

BUFNO=*absexp* (maximum value is 255)
specifies the number of buffers to be acquired automatically by the system during OPEN. If BUFNO is omitted, the system automatically acquires two buffers.

If the GETPOOL macro is used to construct the buffer pool, BUFNO is not required.

Source: BUFNO can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

CYLOFL=*absexp* (maximum value is number of tracks minus 1)
specifies the number of tracks on each cylinder that is reserved as an overflow area. The overflow area contains records forced off prime area tracks when additional records are added to the prime area track in ascending key sequence. ISAM maintains pointers to records in the overflow area so that the entire data set is logically in ascending key sequence.Tracks in the cylinder overflow area are used by the system only if OPTCD=Y is specified. For a more complete description of cylinder overflow area, refer to the space allocation section of *z/OS DFSMS Using Data Sets*.

Source: When an indexed sequential data set is allocated, CYLOFL can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. When an existing indexed sequential data set is processed, CYLOFL should be omitted, allowing the data set label to supply the parameter.

DDNAME=*symbol*
specifies the name that is used to identify the job control language data definition (DD) statement that defines the indexed sequential data set being allocated or processed.

Source: DDNAME can be supplied in the DCB macro or by the problem program before an OPEN macro is issued to open the data set.

DSORG={IS|ISU}
specifies the data set organization, and whether the data set contains any location-dependent information that would make it unmovable. You can specify:

IS
specifies an indexed sequential data set organization.

ISU
specifies an indexed sequential data set that contains location-dependent information. You can specify **ISU** only when creating an indexed sequential data set.

Source: DSORG must be specified in the DCB macro. When an indexed sequential data set is allocated, DSORG=IS or ISU must also be specified in the DCB subparameter of the corresponding DD statement.

EODAD=*relexp*
specifies the address of the routine given control when the end of an input data set is reached. For ISAM, this parameter applies only to scan mode when a data set is open for an input operation. Control is given to this routine when a GET macro is issued and there are no more input records to retrieve. For additional information on the EODAD routine, see *z/OS DFSMS Using Data Sets*.

Source: EODAD can be supplied in the DCB macro or by the problem program before the end of the data set is reached.

EXLST=*relexp*
specifies the address of the DCB exit list. EXLST is required only if the problem program uses the data control block OPEN exit routine for additional processing.

For the functions, format, and requirements for exit list processing, see *z/OS DFSMS Using Data Sets*. The exit list must reside below the line.

Source: EXLST can be supplied in the DCB macro or by the problem program before the relevant function is needed.

KEYLEN=*absexp* (maximum value is 255)
specifies the length, in bytes, of the key associated with each record in an indexed sequential data set. When blocked records are used, the key of the last record in the block (highest key) is used to identify the block. However, each logical record in the block has its own identifying key that ISAM uses to access a given logical record.

Source: When an indexed sequential data set is allocated, KEYLEN can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. When an existing indexed sequential data set is processed, KEYLEN must be omitted, allowing the data set level to supply the *key length* value. KEYLEN=0 is not valid for an indexed sequential data set.

LRECL=*absexp* (maximum value is device-dependent)
specifies the length, in bytes, for fixed-length records, or it specifies the maximum length, in bytes, for variable-length records. The value specified in LRECL cannot exceed the value specified in BLKSIZE. When fixed, unblocked records are used and the relative key position (as specified in the RKP parameter) is zero, the value specified in LRECL should include only the data length (the key is not written as part of the fixed, unblocked record when RKP=0).

Source: When an indexed sequential data set is allocated, LRECL can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. When an existing indexed sequential data set is processed, LRECL must be omitted, allowing the data set label to supply the value.

MACRF={{(PM)}
{(PL)}
{(GM[,S{K|I}])}
{(GL[,S{K|I}][,PU])}}
specifies the type of macros, the transmittal mode, and type of search that are used with the data set being processed. The parameter can be coded in any of the combinations shown above. You can specify the following characters for QISAM:

The following characters can be specified only when the data set is being created (load mode) or additional records are being added to the end of the data set (resume load):

PL
specifies that PUT macros are used in the locate transmittal mode. The system provides the problem program with the address of a buffer containing the data to be written into the data set.

PM
specifies that PUT macros are used in the move transmittal mode. The system moves the data to be written from the problem program work area to the buffer being used.

The following characters can be specified only when the data set is being processed (scan mode) or when records in an indexed sequential data set are being updated in place:

GL
specifies that GET macros are used in the locate transmittal mode. The system provides the problem program with the address of a buffer containing the logical record read.

GM
specifies that GET macros are used in the move mode. The system moves the logical record from the buffer to the problem program work area.

I
specifies that actual device addresses (MBBCCHHR) are used to search for a record (or the first record) to be read.

K
specifies that a key or key class is used to search for a record (or the first record) to be read.

PU
specifies that PUTX macros are used to return updated records to the data set.

S
specifies that SETL macros are used to set the beginning location for processing the data set.

Source: MACRF must be coded in the DCB macro.

NTM=*absexp* (maximum value is 99)
specifies the number of tracks that are created in a cylinder index before a higher-level index is created. If the cylinder index exceeds this number, a master index is created by the system. If a master index exceeds this number, the next level of master index is created. The system creates as many as three levels of master indexes. NTM is ignored unless the master index option (OPTCD=M) is selected.

Source: When an indexed sequential data set is being allocated, NTM can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. When an indexed sequential data set is being processed, master index information is supplied to the data control block from the data set label, and NTM must be omitted.

OPTCD={[I][L][M][R][U][W][Y]}
specifies the optional services that are performed by the system when an indexed sequential data set is being allocated or updated. You can code the following characters in any order, in any combination, and without commas between characters:

I
specifies that the system uses the independent overflow areas to contain overflow records. It is only the use of the allocated independent overflow area that is optional. Under certain conditions, the system designates an overflow area that was not allocated for independent overflow by the problem program. *z/OS DFSMS Using Data Sets* explains how to allocate space for an indexed sequential data set.

L
specifies that the data set is to contain records flagged for deletion. A record is flagged for deletion by placing a hexadecimal value of 'FF' in the first data byte. Records flagged for deletion remain in the data set until the space is required for another record to be added to the track and are ignored during sequential retrieval of the indexed sequential data set (QISAM, scan mode). This option cannot be specified for blocked fixed-length records if the relative key position is

0 (RKP=0), or it cannot be specified for variable-length records if the relative key position is 4 (RKP=4).

When an indexed sequential data set is being processed with BISAM interface to VSAM, a record with a duplicate key can be added to the data set (WRITE KN macro), only when OPTCD=L is specified and the original record (the one whose key is being duplicated) is flagged for deletion.

M
specifies that the system create and maintain a master index or indexes according to the number of tracks specified in NTM.

R
specifies that the system place reorganization statistics in the data control block. The problem program can analyze these statistics to determine when to reorganize the data set. If OPTCD is omitted, the reorganization statistics are automatically provided. However, if you use OPTCD, you must specify OPTCD=R to obtain the reorganization statistics.

U
specifies that the system is to accumulate track index entries in storage and write them as a group for each track of the track index. OPTCD=U can be specified only for fixed-length records. The entries are written in fixed-length unblocked format.

W
specifies a validity check on each record that is written.

Y
specifies that the system is to use the cylinder overflow areas to contain overflow records. If OPTCD=Y is specified, CYLOFL specifies the number of tracks used for the cylinder overflow area. The reserved cylinder overflow area is not used unless OPTCD=Y is specified.

Source: When an indexed sequential data set is allocated, OPTCD can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before an OPEN macro is issued to open the data set. However, all optional services must be requested from the same source. When an existing indexed sequential data set is processed, the optional service information is supplied to the data control block from the data set label, and OPTCD should be omitted.

RECFM={V[B]|F[B]}
specifies the format and characteristics of the records in the data set. If the RECFM parameter is omitted, variable-length records (unblocked) are assumed. You can specify:

B
specifies that the data set contains blocked records.

F
specifies that the data set contains fixed-length records.

V
specifies that the data set contains variable-length records.

Source: When an indexed sequential data set is allocated, RECFM can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before an OPEN macro is issued to open the data set. When an existing indexed sequential data set is processed, the record format information is supplied by the data set label, and RECFM should be omitted.

If the record format information is supplied in the DD statement or the DCB, it must agree with the information in the data set label.

RKP=*absexp*
specifies the relative position of the first byte of the key within each logical record. For example, if RKP=9 is specified, the key starts in the 10th byte of the record. Do not specify the delete option (OPTCD=L) if the relative key position is the first byte of a blocked fixed-length record or the fifth byte of a variable-length record. If the RKP parameter is omitted, RKP=0 is assumed.

If unblocked fixed-length records with RKP=0 are used, the key is not written as a part of the data record, and the delete option can be specified. If blocked fixed-length records are used, the key is

written as part of each data record; either RKP must be greater than zero or the delete option must not be used.

If variable-length records (blocked or unblocked) are used, and if the delete option is not specified, RKP must be 4 or greater. If the delete option is specified, RKP must be specified as 5 or greater. The 4 additional bytes allow for the block descriptor word in variable-length records.

Source: When an indexed sequential data set is allocated, RKP can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. When an existing indexed sequential data set is processed, the RKP information is supplied by the data set label and the RKP parameter should be omitted.

SYNAD=*relexp*
specifies the address of the error analysis routine given control when an uncorrectable input/output error occurs. The entry point of this SYNAD routine must reside below the line. The contents of the registers when the error analysis routine is given control are described in *z/OS DFSMS Using Data Sets*. Additional status information available to the SYNAD routine is described in "Status information following an input/output operation" on page 373.

The error analysis routine must not use the save area pointed to by register 13. The system does not restore registers when it regains control from the error analysis routine. The error analysis routine can issue a RETURN macro that uses the address in register 14 to return control to the system.When control is returned in this manner, the system returns control to the problem program and proceeds as though no error had been found; if the error analysis routine continues processing, the results might be unpredictable.

For additional information on error analysis routine processing for indexed sequential data sets, see *z/OS DFSMS Using Data Sets* .

Source: SYNAD can be supplied in the DCB macro or by the problem program. The problem program can also change the error analysis routine address at any time.

DCB—Construct a data control block (QSAM)

The data control block for a queued sequential access method (QSAM) data set is constructed during assembly of the problem program. You must code DSORG and MACRF in the DCB macro, but the other DCB parameters can be supplied to the data control block from other sources. Each DCB parameter description contains a heading, "Source". The information under this heading describes the sources that can supply the parameter. Each reference to a DCB OPEN exit routine applies also to a JFCBE exit routine.

You can assemble the DCB macro into a program that resides above the 16 MB line, but the program must move it below the line before using it. Except for the DCBE, all areas that the DCB refers to, such as EXLST and EODAD, must be below the 16 MB line.

The format of the DCB macro for QSAM is:

[*label*]	DCB	[BFALN={F|D}] [,BFTEK={S|A}] [,BLKSIZE=*absexp*] [,BUFCB=*relexp*] [,BUFL=*absexp*] [,BUFNO=*absexp*] [,BUFOFF={*absexp*|L}] [,DCBE=*relexp*] "1" on page 213 [,DDNAME=*symbol*] "1" on page 213 [,DEVD={{DA} {TA [,DEN={1|2|3|4}] [,TRTCH={C|E|ET|T}|{COMP|NOCOMP}]} {PR [,PRTSP={0|1|2|3}]} {PC [,MODE=[C|E][R]] [,STACK={1|2}] [,FUNC={I|P|PW[XT]|R|RP[D]| RW[T]|RWP[XT][D]|W[T]}]} {RD [,MODE=[C|E][O|R]] [,STACK={1|2}] [,FUNC={I|P|PW[XT]|R|RP[D]| RW[T]|RWP[XT][D]|W[T]}]}}],DSORG={PS|PSU} [,EODAD=*relexp*] [,EROPT={ACC|SKP|ABE}] [,EXLST=*relexp*] [,LRECL={*absexp*|X|0K|*nnnnn*K}] ,MACRF={{(G{M|L|D}[C])} {(P{M|L|D}[C])} {(G{M|L|D}[C],P{M|L|D}[C])}}[,OPTCD={{B} {T} {U[C]} {C[T][B][U]} {H[Z][B]} {J[C][U]} {W[C][T][B][U]} {Z[C][T][B][U]} {Q[C][B][T]}] {Z}}] [,RECFM={{U[T][A|M]} {V[B][S][T][A|M]} {D[B][S][A]} {F[B|S|T|BS|BT][A|M]}}] [,SYNAD=*relexp*]

Note:

1. This parameter must be supplied before an OPEN macro is issued for this DCB; it cannot be supplied in the open exit routine.

Recommendation: When creating a DCB to open a data set allocated to an SMS-managed volume, do not specify values that would change the data set to a type which cannot be SMS-managed, such as DSORG=PSU.

QSAM supports the following DCB parameters:

BFALN={F|D}
specifies the boundary alignment of each buffer in the buffer pool when the buffer pool is constructed automatically or by a GETPOOL macro. If BFALN is omitted, the system provides doubleword alignment for each buffer.

If the data set being allocated or processed contains ASCII tape records with a block prefix, the block prefix is entered at the beginning of the buffer. Also, data alignment depends on the length of the block prefix.For a description of how to specify the block prefix length, see the description of BUFOFF.

You can specify:

F
specifies that each buffer is on a fullword boundary that is not also a doubleword boundary.

D
specifies that each buffer is on a doubleword boundary.

If the BUILD macro is used to construct the buffer pool, the problem program must control buffer alignment.

Source: BFALN can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. If both BFALN and BFTEK are specified, they must be supplied from the same source.

BFTEK={S|A}
specifies the buffering technique. If BFTEK is omitted, simple buffering is assumed. You can specify:

S
specifies that simple buffering is used.

A
specifies that a logical record interface is used for variable-length spanned records. When BFTEK=A is specified, the open routine acquires a record area equal to the length specified in the LRECL field plus 32 additional bytes for control information. LRECL=0 is invalid. The LRECL provided at open should be the maximum length in bytes. When a logical record interface is requested, the system uses the simple buffering technique.

BFTEK=A is invalid with move transmittal mode.

BFTEK=A is invalid with UNIX files.

To use the simple buffering technique efficiently, you should be familiar with the three transmittal modes for QSAM and the buffering techniques described in *z/OS DFSMS Using Data Sets*.

Source: BFTEK can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine. If both BFTEK and BFALN are specified, they must be supplied from the same source.

BLKSIZE=*absexp* (maximum value is 32760 bytes)
specifies the maximum length, in bytes, of each data block for fixed-length, unblocked records. This parameter specifies the record length. If a physical sequential data set that contains fixed-length records (blocked or unblocked) is accessed with a DCB that has specified a DSORG or undefined, then the BLKSIZE value specified must be a value that is less than or equal to the data sets physical block size in the DSCB.

If the data set for QSAM contains blocks longer than 32760 bytes, then use the BLKSIZE keyword on a DCBE macro and use the DCBE keyword on the DCB macro. That requests the large block interface (LBI). If the system allows LBI, the system modifies the BLKSIZE field in the DCB and your program should not use it.

The actual value that you can specify in BLKSIZE depends on the device type and record format being used. (For additional information about device capacity, refer to the relevant device publication.)

When PDSEs, compressed format data sets, and UNIX files are being processed, the value specified in BLKSIZE can be up to the maximum value. For other data sets on direct access storage devices, the

value specified in BLKSIZE cannot exceed the capacity of a single track.One exception to the device capacity for a logical record is the size of variable-length spanned records. Their length can exceed the value specified in the BLKSIZE parameter (see the description of LRECL).

If fixed-length records are used and you specify a value in BLKSIZE, it must be a whole number multiple of the value specified in LRECL.If the records are unblocked fixed-length records, the value specified in BLKSIZE must equal the value specified in LRECL.

If variable-length records are used, the value specified in BLKSIZE must include the data length (up to 32756 bytes) plus 4 bytes required for the block descriptor word (BDW). For format-D variable-length records, the minimum BLKSIZE value is 18 bytes.

The maximum block size is 32,760 except for ISO/ANSI Version 3 records, where the maximum block size is 2048. As required by the standard, an attempt to exceed 2048 bytes for a Version 3 tape results in a label validation installation exit being called. The exit might allow violation of the standard by writing larger blocks. This restriction does not apply to Version 4 labels. For more information about BLKSIZE restrictions, see *z/OS DFSMS Using Data Sets*.

If ASCII tape records with a block prefix are processed, the value specified in BLKSIZE must also include the length of the block prefix. If an ASCII format DB or DBS tape data set is opened for output using QSAM with the system acquiring the buffers and BUFOFF that is not L specified, the value specified in BLKSIZE must be increased by 4 to allow for a 4 byte QSAM internal processing area. If BUFL is specified, the BUFL value must be increased by 4, instead of the BLKSIZE value.

If variable-length spanned records are used, the value specified in BLKSIZE can be the best one for the device being used or the processing being done. When unit record devices (card or printer) are used, the system assumes records are unblocked. The value specified for BLKSIZE is equivalent to one print line or one card. A logical record that spans several blocks is written one segment at a time.

If undefined-length records are used, the problem program can insert the actual record length into the DCBLRECL field. See the description of LRECL.

Processing PDSEs: The system reblocks PDSE records into its own internal format when the data set is written, and reconstructs the blocks using the block size from the DCB when the data set is read. For fixed-length blocked records, the value specified in BLKSIZE *must* be a multiple of the value in LRECL. The LRECL value must be available to OPEN when the data set is open for output.

For fixed-length unblocked records, LRECL (if specified) must equal BLKSIZE.

When reading a PDSE directory using fixed-length blocked records, you can specify a BLKSIZE of 256 or greater (the LRECL is ignored).

Processing UNIX files: Block boundaries are not maintained within a UNIX file. This means that when you read, records may be distributed among blocks differently than they were written.When BLKSIZE is not specified (by any source), it is defaulted to 80 on input.

System-Determined Block Size: IBM recommends that you not specify block size except in these cases:

- Record format is U.
- Medium is tape without standard labels.
- UNIX file is being processed..

This makes your program less dependent on the physical characteristics of the device.

System-Determined Block Size for DASD Data Sets: For blocked DASD data sets, if the block size is not specified at the time that the data set is created, and the LRECL and RECFM are known, the system derives an optimum block size for the data set. This system-determined block size is retained in the data set label. When the data set is opened for output, OPEN checks the block size in the data set label. If it is a system-determined block size, and the LRECL or RECFM have changed from those specified in the data set label, OPEN redetermines an optimum block size for the data set.

System-Determined Block Size for Tape Data Sets: If you do not specify a block size for a tape data set and the RECFM value is not U, the system determines the optimum block size when the data set is opened for OUTPUT or OUTIN. The system-determined block size depends on the record format and type of the tape data set. See *z/OS DFSMS Using Data Sets* for the table showing the block sizes set for tape data sets.

Source: BLKSIZE can be supplied in the DCB or DCBE macro, in the DCB subparameter of a DD statement, by the problem program before completion of the data control block exit routine, by the data set label of an existing data set, or by the system determining a value for a new data set. The system does not copy BLKSIZE when you code the JCL keyword LIKE. It derives the BLKSIZE from RECFM and LRECL which can be copied. For more information on LIKE, see *z/OS MVS JCL Reference* and *z/OS MVS JCL User's Guide*.

BUFCB=*relexp*
specifies the address of the buffer pool control block that you have constructed by issuing a BUILD or BUILDRCD macro. The buffer pool control block resides below the 16MB line. If the buffer pool is constructed automatically above the line because RMODE31=BUFF is coded on the DCBE macro, omit the BUFCB parameter because the system places the address of the buffer pool control block into the data control block.

If you want the system to acquire buffers automatically above the 16MB line, omit the BUFCB parameter and code RMODE31=BUFF on the DCBE macro. In this case, the buffer pool control block will continue to reside below the 16MB line although the buffers are above the 16MB line.

Source: BUFCB can be supplied in the DCB macro or by the problem program before completion of the data control block exit routine.

BUFL=*absexp* (maximum value is 32760)
specifies the length, in bytes, of each buffer in the buffer pool when the buffer pool is acquired automatically. If BUFL is omitted or if RMODE31=BUFF is coded on the DCBE macro, the system acquires buffers with a length equal to the value specified in BLKSIZE in the DCB (without LBI) or in the DCBE (with LBI). If the problem program requires larger buffers (up to 32760 bytes), BUFL is required. If the data set is for card image mode, BUFL is specified as 160 bytes. The description of DEVD contains a description of card image mode.

If the data set contains ASCII tape records with a block prefix, the value specified in BUFL must also include the length of the block prefix. If an ASCII format DB or DBS tape data set is opened for output using QSAM and BUFOFF that is not L is specified, then the BUFL value, if specified, must be increased by 4 to allow for a 4-byte QSAM internal processing area.

If the buffer pool is constructed by a BUILD, BUILDRCD, or GETPOOL macro, BUFL is not required.

Source: BUFL can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

BUFNO=*absexp* (maximum value is 255)
specifies the number of buffers that are acquired automatically during OPEN. If chained scheduling is specified, the value of BUFNO also determines the maximum number of channel program segments that can be chained and must be specified as more than one. If BUFNO is omitted and the buffers are acquired automatically, the system acquires:

- 1 for a PDSE member
- 1 for an extended format data set in compressed format
- 1 for a UNIX file
- (2 * number of stripes * number of blocks per track) for an extended format data set if it is not in the compressed format
- 2 if the block size is greater than or equal to 32768
- 3 for an IBM 2540 card reader or card punch
- 5 for other types of devices or data sets

It is not useful to specify more than one buffer for a data set in compressed format or a UNIX file unless you expect to reuse the buffer pool for a different data set.

If the buffer pool is constructed by a GETPOOL macro, BUFNO is not required.

Source: BUFNO can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

BUFOFF={*absexp*|L}
specifies the length, in bytes, of the block prefix used with an ASCII tape data set or a tape data set with CCSID. When QSAM is used to read this kind of tape data set, only the data portion (or its address) is passed to the problem program; the block prefix is not available to the problem program. Block prefixes (except BUFOFF=L) cannot be included in QSAM output records. You can specify:

absexp
specifies the length, in bytes, of the block prefix. This value can be from 0 to 99 for an input data set. The value must be 0 for writing an output data set with fixed-length or undefined-length records.

L
specifies that the block prefix is 4 bytes long and contains the block length. BUFOFF=L is used when format-D records are processed. QSAM uses the 4 bytes as a block-descriptor word (BDW). See *z/OS DFSMS Using Data Sets* for further information on format-D records.

Source: BUFOFF can be supplied in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program before an OPEN macro is issued to open the data set. BUFOFF=*absexp* can also be supplied by the second system label of an existing data set; BUFOFF=L cannot be supplied by the label of an existing data set.

DCBE=*relexp*
specifies the address of a DCB Extension (DCBE). The DCBE may reside above the 16MB line. You may assemble a DCB and DCBE in a program that resides above the line if the DCB is copied below the line before opening the copy.

If the DCBE is specified, it must be specified before issuing the OPEN macro. Like the DCB, the DCBE must exist until the data set is closed. Otherwise, there may be unpredictable results.

Only one open DCB at a time can refer to a particular DCBE. After a DCB is successfully closed, you can open a different DCB referring to the DCBE.

The DCBE is not required with QSAM unless the data set requires a DCBE option or if you choose to use DCBE options.

If a DCBE exists, the flags DCBH0 and DCBH1 are both set on. The pointer to the DCBE is stored at offset +0 in the DCB (and replaces the field DCBRELAD). If a DCBE exists, data that would be stored at DCBRELAD is stored in the DCBE (DCBERELA). If a DCBE does not exist, DCBRELAD continues to be located at offset +0 in the DCB.

Source: You can supply the DCBE address in the DCB macro or before issuing an OPEN macro to open the data set.

DDNAME=*symbol*
specifies the name that is used to identify the job control language data definition (DD) statement that defines the data set being allocated or processed.

Source: DDNAME can be supplied in the DCB macro or by the problem program before an OPEN macro is issued to open the data set.

DEVD={DA|TA|PR|PC|RD}[,*options*]
specifies the device type where the data set can or does reside. The device types above are shown with the optional parameters that can be coded when a particular device is used. The devices are listed in order of device independence. For example, if you code DEVD=DA in a DCB macro (or omit DEVD, which causes a default to DA), you can use later the data control block constructed during assembly for any of the other devices. But, if you code DEVD=RD, you can use the data control block

only with a card reader or card reader punch. Unless you are certain that device interchangeability is not required, you should either code DEVD=DA or omit the parameter and allow it to default to DA.

Rule: If the data set can or does reside on DASD, do not code a value other than DEVD=DA.

For spooled data sets, the system ignores these device-dependent parameters. If you code DEVD=PR, PC, or RD, do not code the DCB macro in the first 16 bytes of addressability for the control section.

DEVD is discussed below according to individual device type:

DEVD=DA
specifies that the data control block can be used for a direct access storage device (or any of the other device types described following DA).

DEVD=TA
[,DEN={1|2|3|4}]
[,TRTCH={C|E|ET|T}|{COMP|NOCOMP}]
specifies that the data control block can be used for a magnetic tape data set (or any of the other device types described following TA). If TA is coded, you can specify the following optional parameters:

DEN={1|2|3|4}
specifies the recording density in the number of bits-per-inch per track as follows:

DEN	7-Track	9-Track	18-Track	36-Track
1	556	N/A	N/A	N/A
2	800	800 (NRZI)[“1” on page 218]	N/A	N/A
3	N/A	1600 (PE)[“2” on page 218]	N/A	N/A
4	N/A	6250 (GCR)[“3” on page 218]	N/A	N/A

Notes:

1. NRZI is for nonreturn-to-zero inverted mode.
2. PE is for phase encoded mode.
3. GCR is for group coded recording mode.

For magnetic tape drives that use cartridges, such as the 3480, only a single density is available and is used by the system for reading and writing; any density with the DEN parameter is ignored.

TRTCH={C|E|ET|T}|{COMP|NOCOMP}
The TRTCH parameter has two different sets of values. One of the sets, {C|E|ET|T}, is used to specify the recording technique for 7-track tape. The other set, {COMP|NOCOMP}, is used to specify the recording technique for magnetic tape drives with Improved Data Recording Capability and override the system default.

{C|E|ET|T}
These values specify the recording technique for 7-track tape. One of the above four values can be coded. If TRTCH is omitted, odd parity with no translation or conversion is assumed. You can specify:

C
specifies that the data-conversion feature is used with odd parity and no translation.

E
specifies even parity with no translation or conversion.

ET
specifies even parity with BCDIC to EBCDIC translation required, but no data-conversion feature.

T
specifies that BCDIC to EBCDIC translation is required with odd parity and no data-conversion feature.

{COMP|NOCOMP}
These values specify the recording technique for magnetic tape drives with Improved Data Recording Capability. Either of the two values can be coded. If TRTCH is omitted, the default specified in the active DEVSUPyy member of SYS1.PARMLIB (initially set to NOCOMP) is assumed. You can specify:

COMP
specifies record data in compacted format. COMP is not supported with ISO/ANSI tape labels.

NOCOMP
specifies record data in standard format.

Source: TRTCH can be supplied in the DCB macro, in the DCB subparameter on a DD statement, in the IBM standard tape label or by the problem program before completion of the data control block exit routine.

DEVD=PR
[,PRTSP={0|1|2|3}]
specifies that the data control block is used for a directly allocated printer (or any of the other device types following PR). This has no effect for a spooled (SYSOUT) or subsystem data set. If PR is coded, you can specify the following optional parameter:

PRTSP={0|1|2|3}
specifies the line spacing on the printer. This parameter is not valid if RECFM specifies either machine (RECFM=M), or ANSI or ISO control characters (RECFM=A). If PRTSP is not specified from any source, 1 is assumed. You can specify:

0
specifies that spacing is suppressed (no space).

1
specifies single spacing.

2
specifies double spacing (one blank line between printed lines).

3
specifies triple spacing (two blank lines between printed lines).

Restriction: You cannot use MODE and FUNC subparameters with this specification.

DEVD=PC
[,MODE=[C|E][O|R]]
[,STACK={1|2}]
[,FUNC={I|P|PW[XT]|R|RP [D]|RW[T]| RWP[XT][D]|W[T]}]
specifies that the data control block is used for a card punch (or any of the other device types following PC). If PC is coded, you can specify the following optional parameters:

MODE=[C|E][R]]
specifies the mode of operation for the card punch. If MODE is omitted, E is assumed. You can specify:

C
specifies that the cards are punched in column binary (card image) mode. In column binary mode, the 12 rows in each card column are punched from 2 consecutive bytes of virtual storage. Rows 12 through 3 are punched from the 6 low-order bits of one byte, and rows 4 through 9 are punched from the 6 low-order bits of the following byte.

E
specifies that cards are punched in EBCDIC code.

R
specifies that the program runs in read-column-eliminate mode (3505 card reader or 3525 card punch, read feature).

If you code R for the MODE subparameter of the DCB parameter of the DD statement, you must also code either C or E.

STACK={1|2}
specifies the stacker bin where the card is placed after punching completes. If this parameter is omitted, stacker number 1 is used. You can specify:

1
specifies stacker number 1.

2
specifies stacker number 2.

FUNC={I|P|PW[XT]|R|RP[D]|RW [T]|RWP[XT][D]|W[T]}
specifies the type of 3525 card punch data sets to be used. If FUNC is omitted from all sources, a data set opened for input defaults to read only, and a data set opened for output defaults to punch only. You can specify:

D
specifies that the data protection option is used. The data protection option prevents punching information into card columns that already contain data. When the data protection option is used, an 80-byte data protection image (DPI) must be previously stored in SYS1.IMAGELIB. Specify its name in the FCB parameter of the DD statement. Data protection applies only to the output punch portion of a read and punch or read, punch, and print operation.

I
specifies that the data in the data set is punched into cards and printed on the cards. The first 64 characters are printed on line 1 of the card and the remaining 16 characters are printed on line 3.

P
specifies that the data set is for punching cards. See the description of the character X for associated punch and print data sets.

R
specifies that the data set is for reading cards.

T
specifies that the two-line print option is used. The two-line print option allows two lines of data to be printed on the card (lines 1 and 3). If T is not specified, the multiline print option is used;this allows printing on all 25 possible print lines. In either case, the data printed can be the same as the data punched in the card, or it can be entirely different data.

W
specifies that the data set is for printing. See the description of the character X for associated punch and print data sets.

X
specifies that an associated data set is opened for output for both punching and printing. Coding the character X distinguishes the 3525 printer output data set from the 3525 punch output data set.

DEVD=RD
[,MODE=[C|E][O|R]]
[,STACK={1|2}]
[,FUNC={I|P|PW[XT]|R|RP [D]|RW[T]| RWP[XT][D]|W[T]}]

RD
specifies that the data control block is used with a card reader or card read punch. If RD is specified, the data control block cannot be used with any other device type. When RD is coded, you can specify the following optional parameters:

MODE=[C|E][O|R]
specifies the mode of operation for the card reader. You can specify:

C
specifies that the cards to be read are in card image mode. In card image mode, the 12 rows of each card column are read into 2 consecutive bytes of virtual storage. Rows 12 through 3 are read into the 6 low-order bits of one byte, and rows 4 through 9 are read into the 6 low-order bits of the following byte.

E
specifies that the cards to be read contain data in EBCDIC code.

O
specifies that the program runs in optical mark read mode (3505 card reader).

R
specifies that the program runs in read-column-eliminate mode (3505 card reader and 3525 card punch, read feature).

If the MODE parameter for a 3505 or 3525 is specified in the DCB subparameter of a DD statement, either C or E must be specified if R or 0 is specified.

STACK={1|2}
specifies the stacker bin into which the card is placed after being read. If this parameter is omitted, stacker number 1 is used. You can specify:

1
specifies stacker number 1.

2
specifies stacker number 2.

FUNC={I|P|PW[XT]|R|RP[D]| RW[T]|RWP[XT][D]|W[T]}
defines the type of 3525 card punch data sets used. If the FUNC parameter is omitted from all sources, a data set opened for input defaults to read only, and a data set opened for output defaults to punch only. You can specify:

D
specifies that the data protection option is used. The data protection option prevents punching information into card columns that already contain data. When the data protection option is used, an 80-byte data protection image (DPI) must be previously stored in SYS1.IMAGELIB. Specify its name in the FCB parameter of the DD statement. Data protection applies only to the output punch portion of a read and punch or read, punch, and print operation.

I
specifies that the data in the data set is punched into cards and printed on the cards. The first 64 characters are printed on line 1 of the card and the remaining 16 characters are printed on line 3.

P
specifies that the data set is for punching cards. See the description of the character X for associated punch and print data sets.

R
specifies that the data set is for reading cards.

T
specifies that the two-line option is used. The two-line print option allows two lines of data to be printed on the card (lines 1 and 3). If T is not specified, the multiline print option is used. Thisallows printing on all 25 possible print lines. In either case, the data printed can be the same as the data punched in the card, or it can be entirely different data.

W
specifies that the data set is for printing. See the description of the character X for associated punch and print data sets.

X
specifies that an associated data set is opened for output for both punching and printing. Coding the character X distinguishes the 3525 printer output data set from the 3525 punch output data set.

Source: DEVD can be supplied only in the DCB macro. However, the optional parameters can be supplied in the DCB macro, the DCB subparameter of a DD statement, or by the problem program before completion of the data control block exit routine.

DSORG={PS|PSU}
specifies the data set organization and whether the data set contains any location-dependent information that makes it unmovable. You can specify:

PS
specifies a physical sequential data set.

PSU
specifies a physical sequential data set containing location-dependent information that makes it unmovable.

Restriction: An unmovable data set cannot be SMS-managed. PDSEs cannot be in unmovable data sets. See "NOTE—Provide relative position (BPAM and BSAM—tape and DASD only)" on page 284 for more information about unmovable data sets.

Source: You must code DSORG in the DCB macro.

EODAD=*relexp*
specifies the address of the routine given control when the end of an input data set is reached. Control is given to this routine when a GET macro is issued and there are no additional records to be retrieved. If the record format is RECFM=FS or FBS the end-of-data condition is sensed when a file mark is read or when more data is requested after reading a truncated block.

If the end of the data set is reached but no EODAD address was supplied to the data control block (DCB) or DCBE, or if a GET macro is issued after an end-of-data exit is taken, the task is abnormally terminated. For additional information on the EODAD routine, see *z/OS DFSMS Using Data Sets*.

This end-of-data routine entry point specified in the DCB must reside below the line. If you want the entry point to reside above the line, use the EODAD parameter of the DCBE macro. The EODAD routine (whether it is specified in the DCBE or DCB) receives control in the addressing mode in which the GET macro was issued. See the EODAD parameter description for the DCBE macro, "DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)" on page 230.

Source: EODAD can be supplied in the DCB macro or by the problem program before the end of the data set has been reached.

EROPT={ACC|SKP|ABE}
specifies the action taken by the system if an uncorrectable input/output data validity error occurs and no error analysis (SYNAD) routine address is provided. Or, it specifies the action taken by the system after the error analysis routine has returned control to the system with a RETURN macro. The specified action is taken for input operations for all devices or for output operations to a printer.

Uncorrectable input/output errors resulting from channel operations or direct access operations that make the next record inaccessible cause the task to be abnormally terminated regardless of the action specified in the EROPT parameter.

An example of the next record being inaccessible is where the device sense bits include the 'record not found' condition. This is bit 4 (X'08') in the second device sense byte. Your SYNAD routine can see this byte at +3 in the status area that the access method passes to the SYNAD routine. This byte is device-dependent. This meaning of this bit applies only for DASD.

For UNIX file processing, the system treats EROPT=ACC or EROPT=SKP as EROPT=ABE.

You can specify:

ACC
specifies that the problem program accepts the block causing the error. The system recognizes this option if the DCB is open for INPUT, RDBACK, UPDAT, or OUTPUT (OUTPUT applies to printer data sets only).

SKP
specifies that the block causing the error is skipped. The system tries to process the next block. If it also returns an uncorrectable I/O error, the system again will use the SYNAD and EROPT parameters. The system recognizes SKP if the OPEN macro option was for INPUT, RDBACK, or UPDAT. If the device is a printer, the system treats EROPT=SKP as EROPT=ABE.

ABE
specifies that the error results in the abnormal termination of the task. The system recognizes this option if the DCB is open for INPUT, OUTPUT, RDBACK, or UPDAT. If EROPT is omitted, the ABE action is assumed.

Recommendation: If EROPT is ACC or SKIP, accept or skip processing is done after returning from the error analysis (SYNAD) routine. For this reason, do not issue FEOV from within the error analysis routine.

Source: EROPT can be specified in the DCB macro, in the DCB subparameter of a DD statement, or by the problem program at any time. The problem program can also change the action specified at any time.

EXLST=*relexp*
specifies the address of the DCB exit list. EXLST is required if the problem program requires additional processing for user labels, user totaling, data control block OPEN exit routines, end-of-volume, block count exits, defining a forms control buffer (FCB) image, using the JFCBE exit (for the 3800 printer), or using the DCB ABEND exit for abend condition analysis.

The exit list must reside below the line. For the functions, format, and requirements of exit list processing, see *z/OS DFSMS Using Data Sets*. Exit routines can reside above the 16 MB line if you use the technique described in Figure 4 on page 153.

Source: EXLST can be supplied in the DCB macro or by the problem program any time before the relevant function is needed.

LRECL={*absexp*|X|0K|*nnnnn*K}
specifies the length, in bytes, for fixed-length records. Or, it specifies the maximum length, in bytes, for variable-length or undefined-length (output only) records. The value specified in LRECL cannot exceed the value specified in BLKSIZE except when variable-length spanned records are used.

Unblocked fixed-length records: the value specified in LRECL must be equal to the value specified in BLKSIZE.

Blocked fixed-length records: The value specified in LRECL must be evenly divisible into the value specified in BLKSIZE. LRECL is required for blocked fixed-length records.

Variable-length records: the value specified in LRECL must include the maximum data length (up to 32752 bytes) plus 4 bytes for the record-descriptor word (RDW).

Undefined-length records: the problem program must insert the actual logical record length into the DCBLRECL field before writing the record, or else the maximum-length record is written.

Variable-length spanned records: the logical record length (LRECL) can exceed the value specified in BLKSIZE, and a variable-length spanned record can exceed the maximum block size (32760 bytes). When the logical record length exceeds the maximum block size (for non-XLRI processing), you must specify LRECL=X and use GET or PUT locate mode.

UNIX files: record boundaries are not maintained within a binary UNIX file. When LRECL is not specified (by any source), it is defaulted to 80 on input.

ISO/ANSI/FIPS variable-length spanned records: (RECFM=DS or RECFM=DBS), you can use the extended logical record interface (XLRI) when the maximum logical record length exceeds 32760 bytes.XLRI must be invoked by specifying LRECL=0K or LRECL=nnnnnK.

nnnnnK
specifies the size of the record area (in 1024-byte units) required to contain the longest logical record of the data set. The value nnnnnK can range from 1K to 16383K.

0K
specifies that the length of the longest logical record must come from the DD statement or the data set label. XLRI processing is only valid in QSAM locate mode. You must not specify LRECL=X for RECFM=DS or DBS.

When LRECL=0K is used in the DCB, the LRECL data must come from JCL, the file label (for an input data set), or from the DCB exit during open merge.

X
specifies that the logical record length exceeds the maximum block size (32760 bytes), and GET or PUT locate mode is used.

Source: LRECL can be supplied in the DCB macro, in the DCB subparameter of a DD statement, by the problem program before completion of the data control block exit routine, or by the data set label of an existing data set. The label indicates a logical record length of '99999' when an IBM standard label tape contains a logical record equal to or greater than 100KB. The label indicates '00000' if the same maximum is reached for an ISO/ANSI label tape.

Record length can be derived from the data class associated with the data set. Record length can also be derived from the JCL keyword LIKE. However, if LRECL is specified in the DCB macro, it overrides the value derived from data class or LIKE. For more information, see *z/OS MVS JCL Reference*.

Although LRECL=0K is only valid with RECFM=DS or DBS, you can specify the 0K option on the DCB macro even though the RECFM is not determined until the DCB is opened. (The RECFM is obtained from the data set label or the DD statement.) If you specify neither the DS nor the DBS option, the system turns the 0K indicator off, and restores it when the DCB is closed.

MACRF={{(G{M|L|D}[C])}
{(P{M|L|D}[C])}}
{(G{M|L|D}[C],P{M|L|D }[C])}}
specifies the type of macros (GET, PUT or PUTX, CNTRL, RELSE, and TRUNC) and the transmittal modes (move, locate, and data) that are used with the data set being created or processed. The parameter can be coded in any of the combinations shown above. You can specify:

C
specifies that the CNTRL macro is used with the data set. If you specify C, the device must be one of these described in "CNTRL—Control directly allocated input/output device (BSAM and QSAM)" on page 170. The CNTRL option can be specified with GET in the move mode only.Use of the CNTRL macro is invalid for 3525 input data sets.

D
specifies that the data transmittal mode is used (only the data portion of a record is moved to or from the work area). Data mode is used only with variable-length spanned records.

G
specifies that GET macros are used. Specifying G also provides the routines that allow the problem program to issue RELSE macros. G is required if the OPEN option is INPUT or UPDAT. It has no effect if the OPEN option is OUTPUT or EXTEND.

L
specifies that the locate transmittal mode is used; the system provides the address of the buffer containing the data.

M
specifies that the move transmittal mode is used; the system moves the data from the buffer to the work area in the problem program.

P
specifies that PUT or PUTX macros are used. Specifying P also provides the routines that allow the problem program to issue TRUNC macros. P is required if the OPEN option is OUTPUT or EXTEND. It has no effect if the OPEN option is INPUT. P may be specified if the OPEN option is UPDAT.

Rule: For data sets processed by QSAM using MACRF=(GM) or MACRF=(PM), do not code BFTEK=A.

Source: MACRF can be supplied only in the DCB macro.

OPTCD={{B}
{T}
{U[C]}
{C[T][B][U]}
{H[Z][B]}
{J[C][U]}
{W[C][T][B][U]}
{Z[C][T][B][U]}
{Q[C][B][T]}
{Z}}
specifies the optional services that are used with the sequential data set. Two of the optional services, OPTCD=B and OPTCD=H, cannot be specified in the DCB macro. They are requested in the DCB subparameter of a DD statement. Because all optional services codes must be supplied by the same source, you must omit OPTCD from the DCB macro if either of these options is requested in a DD statement.

Note: If OPTCD=B is specified on the DD statement for a multivolume tape data set, the system will generate the equivalent of individual concatenated DD statements for each volume serial number. This means that the system allocates one tape drive for each volume.

You can code the following characters, in any order, and without commas between characters:

C
specifies that chained scheduling is used. OPTCD=C cannot be specified when either BFTEK=A or BFTEK=R is specified for the same data control block. Also, chained scheduling cannot be specified for associated data sets or printing on a 3525 and is ignored for direct access storage devices.

Note: Except where it is not allowed, chained scheduling is used whether requested or not. For conditions under which it is not allowed, see *z/OS DFSMS Using Data Sets*.

J
specifies that the first data byte in the output data line is a 3800 table reference character.This table reference character selects a particular character arrangement table for the printing of the data line and can be used singly or with ISO/ANSI/FIPS or machine control characters. This option has effect for DASD data sets, SYSOUT data sets, and a directly allocated IBM 3800 Printing Subsystem. On DASD, this indication is saved in the data set label and can be available to programs that read the data. For a partitioned data set, the OPTCD value applies to all members. If the SYSOUT data set is printed on a device that does not support table reference character, the system discards that byte.

Q
requests conversion of the tape records between what is stored on tape and what is supplied from/to the problem program. For input requests, conversion is done after the data is read from tape. For output requests, conversion is done just before the record is written to tape.

The Q option implies that the character representation of the data on tape differs from that seen by the problem program. Data management converts records according to one of the following techniques:

- **CCSID Conversion**

  If CCSIDs are supplied from any source for ISO/ANSI V4 tapes, records are converted between the CCSID which represents the data on tape and the CCSID as seen by the problem program. You can also prevent conversion by supplying a special CCSID.

- **Default Character Conversion**

  If you are using non-ISO/ANSI V4 tapes or if CCSIDs are not supplied by any source, data management converts the records between ASCII code (which represents the data on tape) to EBCDIC code (which is seen by the problem program) using specific tables defined for this default character conversion.

Refer to *z/OS DFSMS Using Data Sets*, for a complete description of CCSID conversion and default character conversion.

Refer to *z/OS DFSMS Using Magnetic Tapes* for more information about ISO/ANSI labels.

Q is supported only for a magnetic tape that does not have IBM standard labels. If the tape has ISO/ANSI/FIPS labels (LABEL=(,AL)), the system assumes OPTCD=Q.

T

requests the user totaling function. If this function is requested, EXLST should specify the address of an exit list that includes a totaling entry. T cannot be specified for a SYSIN or SYSOUT data set.

User totaling can be specified for only sequential data sets that are not extended format data sets. If specified for a partitioned data set, a PDSE, an extended format data set, or a UNIX file,user totaling is ignored.

U

unblocks data checks (permits them to be recognized as errors) and allows analysis by the appropriate error analysis routine (SYNAD exit routine). If the U option is omitted, data checks are not recognized as errors. This option has effect only for a printer with the universal-character-set feature (UCS) or the IBM 3800 Printing Subsystem.

For magnetic tape drives, sets to "tape write immediate" mode.

W

specifies, for DASD, that the system performs a validity check on each block written on a direct access storage device.

OPTCD=W is ignored for PDSEs, extended format data sets, and UNIX files.

The system reads each block back. The intent is to ensure that the data would survive a subsequent power failure.

Recommendation: Because of the performance degradation and the reliability of modern IBM devices and recovery techniques, IBM recommends not coding OPTCD=W.

For buffered tape devices, device end interrupt is given only when a block is physically on the device. By specifying OPTCD=W with buffered devices, you do not benefit from the performance advantage of buffering.

Z

requests for magnetic tape input only, that the system shorten its normal error recovery procedure to consider a data check as a permanent I/O error after five unsuccessful attempts to read a record.OPTCD=Z is used when a tape is known to contain errors and there is no need to process every record. The error analysis routine (SYNAD) should keep a count of permanent errors and terminate processing if the number becomes excessive.

For other devices, the Z option is ignored.

Note: The following optional services can be specified in the DCB subparameter of a DD statement.If either of these options are requested, the complete OPTCD parameter must be supplied in the DD statement.

B

forces the end-of-volume (EOV) routine to disregard the end-of-file recognition for magnetic tape. When this occurs, the EOV routine uses the number of volume serial numbers to determine end of file. For an input data set on a standard labeled (SL or AL) tape, the EOV routine treats EOF labels as EOV labels until the volume serial list is exhausted.After all the volumes have been read,

control is passed to your end-of-data routine. This option allows SL or AL tapes to be read out of volume sequence or to be concatenated to another tape using one DD statement.

H
specifies that the VSE/MVS interchange feature is being used with the data set. It is on magnetic tape and may contain VSE embedded checkpoint records. You cannot use this option with LBI.

Source: OPTCD can be supplied in the DCB macro, in the DCB subparameter of a DD statement, in the data set label for direct access storage devices, or by the problem program before completion of the DCB open exit routine or JFCBE exit routine. However, all optional services must be requested from the same source.

RECFM={{U[T][A|M]}
{V[B][S][T][A|M]}
{D[B][S][A]}
{F[B|S|T|BS|BT][A|M]}}
specifies the record format and characteristics of the data set being allocated or processed. All record formats can be used in QSAM. You can specify:

A
specifies that the records in the data set contain ISO/ANSI control characters. For a description of control characters, see Appendix C, “Control characters,” on page 397.

B
specifies that the data set contains blocked records.

D
specifies that the data set contains variable-length tape records with RDWs in ASCII format. See OPTCD=Q and BUFOFF for a description of how to specify these types of data sets.

F
specifies that the data set contains fixed-length records.

M
specifies that the records in the data set contain machine code control characters. For a description of control characters, see Appendix C, “Control characters,” on page 397. RECFM=M cannot be used with ASCII data sets.

S
specifies, for fixed-length records, that the records are written as standard blocks. Except for the last block or track in the data set, the data set does not contain any truncated blocks or unfilled tracks. Do not code S to retrieve fixed-length records from a data set that was allocated using a RECFM other than standard.

For variable-length records, S specifies that a record can span more than one block.

T
specifies that track overflow is used with the data set. Track overflow allows a record to be written partially on one track and the remainder of the record on the following track (if required).

Note: This is an obsolete option. The system ignores it.

U
specifies that the data set contains undefined-length records.

Restriction: Format-U records are not supported for Version 3 or Version 4 ISO/ANSI tapes. An attempt to process a format-U record for a Version 3 or Version 4 tape results in a label validation installation exit being called.

On ISO/ANSI Version 1 (ISO 1001-1969 or ANSI X3.27-1969) tapes, format-U records can be used for input only. These records are the same as in other types of data sets except that any control characters must be ISO/ANSI control characters and block prefixes can be used.

V
specifies that the data set contains variable-length records.

Restrictions:

- Do not specify RECFM=FS or RECFM=FBS for a partitioned data set or PDSE, because it will cause an abend.
- RECFM=V cannot be specified for a card reader data set or an ISO/ANSI tape data set.
- RECFM=VS, VBS, DS, or DBS cannot be specified for a SYSIN data set.
- RECFM=VS or VBS cannot be specified for a UNIX file.
- RECFM=DS or RECFM=DBS provides blocking, unblocking, and segmenting for Version 3 ISO/ANSI tape data sets.

Source: RECFM can be supplied in the DCB macro, in the DCB subparameter of a DD statement, by the problem program before completion of the data control block exit routine, or by the data set label of an existing data set.

Record format can be derived from the data class associated with the data set. Record format can also be derived from the JCL keyword LIKE. However, if RECFM is specified in the DCB macro, it overrides the value derived from data class or LIKE. For more information, see *z/OS MVS JCL Reference*.

SYNAD=*relexp*

specifies the address of the error analysis (SYNAD) routine given control if an uncorrectable input/output error occurs. The entry point of this SYNAD routine must reside below the line. If you wish the entry point to reside above the line, use the SYNAD parameter of the DCBE macro. You can also use the technique shown in Figure 4 on page 153. The contents of the registers when the error analysis routine is given control are described in *z/OS DFSMS Using Data Sets*. Additional status information available to the SYNAD routine is described in "Status information following an input/output operation" on page 373.

The system detects I/O errors asynchronously but calls your SYNAD routine synchronously when you issue a GET macro for the failed block or when you issue a PUT macro that requires the buffer containing the failed block.

The error analysis routine must not use the save area pointed to by register 13. The system does not restore registers when it regains control from the error analysis routine. The error analysis routine can issue a RETURN macro that uses the address in register 14 to return control to the system.

If the error analysis routine returns and the error condition was the result of a data-validity error, the control program takes the action specified in the EROPT parameter; otherwise, the task is abnormally terminated. The control program takes these actions when SYNAD is omitted in the DCB and DCBE or when the error analysis routine returns control.

The SYNAD routine (whether specified in the DCBE or DCB) receives control in the addressing mode in which the GET or PUT macro was issued. On return from the SYNADAF or SYNADRLS macro issued in the SYNAD routine, the high order byte of register 15 will be unpredictable. Therefore, callers of SYNADAF or SYNADRLS in 31-bit addressing mode must either not use register 15 as a base register or restore the high order byte on return from SYNADAF or SYNADRLS.

When operating a directly allocated IBM 3800 Model 3, 6, or 8 using all-points addressability, the SYNAD exit routine is entered if Print Services Facility (PSF) detects an unrecoverable error. However, no error information is available to the SYNAD routine. If you want to continue processing, you must close and reopen the data set to restart PSF.

Source: SYNAD can be supplied in the DCB macro or by the problem program. The problem program can also change the error routine address at any time.

DCBD—Provide symbolic reference to data control blocks (BDAM, BISAM, BPAM, BSAM, QISAM, and QSAM)

The DCBD macro generates a dummy control section that provides symbolic names for the fields in one or more data control blocks. The DCBD macro maps the assembler version of the DCB. Symbols generated by the DCBD macro include some that are not part of the intended programming interface. The names and attributes of the general-use fields appear as part of the description of each data control block in "Data control block symbolic field names" on page 374. Attributes of the symbolically named fields in the

dummy section are the same as the fields in the data control blocks, except for fields containing 3-byte addresses. The symbolically named fields containing 3-byte addresses have length attributes of 4 and are aligned on fullword boundaries.

The symbols generated by the DCBD macro should not be defined in your user program. The symbols are structured as DCBxxxxx, where DCB is the first 3 characters and xxxxx is one or more alphanumeric characters.

The name of the dummy control section generated by a DCBD macro is IHADCB. A USING instruction specifying IHADCB and a dummy section base register must precede the symbolic names in the dummy section. The dummy section base register contains the address of the actual data control block. You can issue the DCBD macro only once in any assembled module. However, you can use the resulting symbolic names for any number of data control blocks by changing the address in the dummy section base register. You can code the DCBD macro at any point in a control section. However, if it is coded at any point other than at the end of a control section, you must code a CSECT instruction to resume the control section.

The format of the DCBD macro is:

b	DCBD	[DSORG=({GS|(*dsorglist*)})] [,DEVD=(*devlist*)]

DSORG=({GS|(*dsorglist*)})
specifies the types of data control blocks for which symbolic names are provided. If the DSORG parameter is omitted, the DEVD parameter is ignored, and symbolic names are provided only for the 'foundation block' portion that is common to all data control blocks.

GS
specifies a data control block for graphics. This parameter cannot be used in combination with any of the below.

dsorglist
You can specify one or more of the following values (each value must be separated by a comma):

BS
specifies a data control block for BSAM.

DA
specifies a data control block for BDAM. Although this option is supported, its use is not recommended.

IS
specifies a data control block for BISAM and QISAM. Although this option is supported, its use is not recommended.

LR
specifies a dummy section for the logical record length field (DCBLRECL) only.

PO
specifies a data control block for BPAM.

PS
specifies a data control block for BSAM and QSAM. PS includes both BS and QS.

QS
specifies a data control block for QSAM.

DEVD=(*devlist*)
specifies the types of devices on which the data set can reside. If DEVD is omitted and BS, QS, or PS is specified in DSORG, symbolic names are provided for all the device types listed below.

devlist
You can specify one or more of the following values (each value must be separated by a comma). If you specify more than one value, they must have parentheses around them.

DA
Direct access storage device

PC
Directly-allocated card punch (not SYSOUT)

PR
Directly-allocated printer (not SYSOUT)

RD
Directly-allocated card reader or read punch feed (not spooled)

TA
Magnetic tape

MR
Magnetic character reader

DCBE—(BDAM, BSAM, QSAM, BPAM, and EXCP)

The DCB extension (DCBE) provides functions that augment those provided by the DCB. A DCBE is optional. The DCBE must reside in storage that you can access and modify. This storage might be located above or below the 16 MB line independently of whether your program is executing in 31-bit addressing mode. The DCBE is specified using the DCBE parameter of the DCB macro. The DCBE must be in the same storage key as the corresponding DCB. If they are not in the same storage key, the OPEN fails with message IEC190I.

The DCBE must not be shared by multiple DCBs that are open. After the DCB is successfully closed, the user might open a different DCB pointing to the same DCBE. Your program might refer to DCBE fields symbolically by using the IHADCBE mapping macro and the DCBDCBE address in the DCB (using the DCBD mapping macro).

OPEN sets a flag (DCBEMD31) in the DCBE if 31-bit SAM is supported. You might test the DCBEMD31 flag during the DCB OPEN exit routine or anytime until CLOSE. In a concatenation, if you turned on the DCB unlike attributes bit before using OPEN, then OPEN sets DCBEMD31 on if the current access supports data above the line. If you did not turn on the DCB unlike attributes bit, then OPEN sets DCBEMD31 on if all the data sets in the concatenation support data above the line. Otherwise, OPEN sets DCBEMD31 off.

The purpose of this test is to allow you to determine that the SAM 31-bit interfaces do not work for the data set being opened. DCBEMD31 also remains off on a DFP level that supports none of the SAM 31-bit interfaces.

The value of DCBEMD31 does not specify whether an OPEN or CLOSE issuer or parameter list might be the 31-bit type.

Each DCBE parameter description contains a heading, "Source." The information under this heading describes when you might set the parameter.

The format of the DCBE macro is:

[label]	DCBE	[,BLKSIZE=*n*] [,BLOCKTOKENSIZE={LARGE|SMALL}] [,BYPASS_AUTH={NO|YES}][,BYPASS_EXTENT_CHECK={NO|YES}] [,CAPACITYMODE=*XCAP*] [,CONCURRENTRW=({YES[,{EXTLOCK|TRKLOCK}]|NO})] [,DSENCRYPT={OK|NOTOK}] [,EADSCB=OK|NOTOK] [,EODAD=*relexp*] [,FIXED=USER] [,GETSIZE={YES|NO}] [,HYPERWRITE={YES|NO}] [,LOC={ANY|BELOW}] [,MULTACC=*n*] [,MULTSDN=*n*] [,NOVER={YES|NO}] [,PASTEOD={YES|NO}] [,RMODE31={BUFF|NONE}] [,SYNAD=*relexp*] [,SYNC={SYSTEM|NONE|(NUMFILES,*nnn*)}] [,VERSION={0|1}]

Note:

1. With BDAM only the EADSCB operand has an effect. IBM recommends that you not code other operands for BDAM.
2. With EXCP only the BLKSIZE, BLOCKTOKENSIZE, CAPACITYMODE, EADSCB, LOC, and SYNC operands have an effect. IBM recommends that you not code other operands for EXCP.

BLKSIZE=*n*
requests the large block interface (LBI) even if you code 0. If the value is nonzero, this parameter specifies the maximum block length in bytes. For fixed-length, unblocked records, a nonzero value specifies the record length. The actual value that you can specify in BLKSIZE depends on the device type and the record format being used. Most of the LBI information is described in this documentation. For more information, see *z/OS DFSMS Using Data Sets*.

To process a large (greater than 32760) block size tape with BSAM or QSAM, your program must provide a DCBE along with the DCB for the data set. The DCBE must indicate that the application can process large block sizes by specifying the BLKSIZE parameter. If you specify BLKSIZE=0 on the DCBE macro, the open function sets the DCBE block size field from the first nonzero value from one of the following:

- The JCL parameters DCB=BLKSIZE or BLKSIZE.
- The data set label on tape or disk.
- Determined by the system when the following conditions are met:
  - The OPEN option is OUTPUT or OUTINX.
  - The record format is fixed or variable.
  - The maximum logical record length is available.

  The upper limit to the block size that is determined by OPEN is the BLKSZLIM value on the DD statement or a limit that is set in the data class or in SYS1.PARMLIB by a system programmer. See *z/OS DFSMS Using Data Sets* for a description of the system-determined block size function.

If you code a value for DCBE BLKSIZE, even 0, and the large block interface is used, the system uses the DCB BLKSIZE field for scratch purposes.

If you do not code a value for DCBE BLKSIZE but there is a DCB BLKSIZE value that is specified, the DCB BLKSIZE applies.

BLOCKTOKENSIZE={LARGE|SMALL}

This option allows you to specify whether your application program can handle the interface for large format data sets. The default is SMALL.

The IGDSMSxx member of the SYS1.PARMLIB system data set has an option that affects whether your program can open large format data sets. Only a system programmer can change the IGDSMSxx member.

You can ask your system programmer which of the following two values of the option are in effect:

- BLOCKTOKENSIZE(REQUIRE):

  It means that all programs that open any large format data set must have BLOCKTOKENSIZE=LARGE on the DCBE macro unless the OPEN option is INPUT. Also, the data set contains no more than 65535 tracks on the volume. Otherwise, OPEN issues an ABEND macro.
- BLOCKTOKENSIZE(NOREQUIRE):

  This is the default when IBM supplies the operating system. This means that when programs open a large format data set, they do not need to code BLOCKTOKENSIZE=LARGE on the DCBE unless one of these is true:

  – The DCB signifies BSAM with the NOTE or POINT macro. This means that the DCB has a MACRF value of RP or WP or both.

  – The DCB signifies EXCP (MACRF=E).

When you code BLOCKTOKENSIZE=LARGE on the DCBE macro, it has the following effects:

- The DCBELARGE bit in DCBEFLAG3 of the DCBE is set to 1.
- Your program can use this DCBE option to open large format data sets without regard to the value of the BLOCKTOKENSIZE option in the IGDSMSxx member of the SYS1.PARMLIB system data set. It means that your program is aware of the differences in the DSCB (DS1TTTHI), DEB (DEBTTTHI), and TTR conversion routines. These pics are described in *z/OS DFSMSdfp Advanced Services*.
- If you use BSAM with the NOTE and POINT macros, then the OPEN macro allows the opening of large format data sets. You also can use BPAM with large block tokens. Your program is signifying that the NOTE and POINT macros use four-byte block tokens (or addresses) instead of three-byte block tokens. When using BPAM with a PDS, this does not affect the TTR values used for BLDL and FIND. For more details, see the descriptions of the NOTE and POINT macros. Also, with BPAM the content of DCBRELAD and DCBERELA, which identify the location of the beginning of the member, are unaffected.

  Note: If you pass to POINT a TTR returned from BLDL, you must right align the three bytes returned into the four byte field that you pass.
- If the DCB macro signifies EXCP, then the OPEN macro allows the opening of large format data sets.

BYPASS_AUTH={NO|YES}

This option specifies whether OPEN should bypass SAF security checking for a DASD sequential, direct, PDS, or PDSE data set. YES means to bypass SAF security. When you code BYPASS_AUTH=YES, the caller must be supervisor state, in system key, or APF-authorized when invoking the OPEN macro instruction.

Source: Set BYPASS_AUTH in the DCBE macro before issuing OPEN. It remains in effect until completion of CLOSE. The default is NO. Specifying any valid value for BYPASS_AUTH results in a DCB version 1 expansion.

BYPASS_EXTENT_CHECK={NO|YES}

This option specifies the user request for Bypass Extent Check, which might result in improved performance when multiple applications are simultaneously reading or writing the same disk extent ranges. Caller must be APF authorized, running in supervisor state or system key. If the associated data set is not an extended format data set, this parameter is ignored.

Normally when your program writes on DASD, the DASD subsystem performs extent serialization that prevents all access to that extent from all other programs on all systems while data is being written. This prevents other programs from reading and writing within the extent and seeing certain types of incomplete updates. If you know that other programs might be reading the data set at the same time that your program is writing, and a mechanism exists within those other programs to avoid or tolerate potentially incomplete updates within an extent, then coding BYPASS_EXTENT_CHECK=YES might give improved performance. This improved performance is due to the DASD subsystem serializing access to only one track at a time, not to the whole extent. No matter what you code for the BYPASS_EXTENT_CHECK keyword, the program always reads consistent data within each track. An application should specify BYPASS_EXTENT_CHECK=YES only if it can tolerate inconsistent data because of concurrent writes to other tracks.

CAPACITYMODE=XCAP

This option specifies that you want to use extended capacity with the NOTE and POINT macros if the device is capable of it. This affects the NOTE and POINT macros when TYPE=ABS is coded, but when TYPE=REL is coded or defaulted. It also allows more blocks on the tape with BSAM, QSAM, and EXCP.

This option has an effect only if the device is an IBM 3590 that is emulating an IBM 3490, and has the right level of maintenance. You might want to request the extended capacity mode for the following reasons:

- It allows BSAM, QSAM, or EXCP programs to read or write longer tapes.
- The BSAM or EXCP program can issue the NOTE and POINT macros with extended capacity mode.

To help your program adapt to various devices and levels of the system, the DCBE has a bit named DCBE_32BIT_INUSE. See the DCBE field descriptions in “Data control block extension (DCBE)” on page 393. After a successful OPEN macro, your code can test this bit. If it is 1, your program knows that one of the following conditions is true:

- The current volume is on a device that supports extended capacity all the time. This bit is on even if you do not code CAPACITYMODE.
- The current volume is on a drive, such as the IBM 3590, that is emulating a lower capacity device, but it is operating in high-capacity mode. The bit is on only if you specify CAPACITYMODE=XCAP.

If you request CAPACITYMODE=XCAP, but your program finds that the DCBE_32BIT_INUSE bit is off, one of the following conditions is true:

- The operating system is down-level and does not support CAPACITYMODE=XCAP.
- The operating system is up-level, but the device does not support extended capacity.

Each volume in a data set and each data set in a sequential concatenation might have differing values for this bit.

A QSAM program is automatically able to read a long tape without requesting extended capacity. A BSAM or EXCP reading program is not able to read a long tape without requesting CAPACITYMODE=XCAP. This is because the NOTE macro with TYPE=ABS returns a larger value than would be returned from lower capacity models such as the 3480 and 3490.

After the drive is switched to high-capacity mode, it remains in that mode if the tape is mounted, even if your program closes the data set and opens the same or a different data set on the tape and does not code CAPACITYMODE.

CONCURRENTRW=({YES[,{EXTLOCK|TRKLOCK}]|NO})

Errors such as ‘invalid track format’ or ‘no record found’ are errors that can occur when a data set is being read at the same time it is being written to. When these errors occur, the system logs them. For extended format data sets you can specify CONCURRENTRW=YES to prevent the system from logging I/O errors that might result from reads that are concurrent with writes.

CONCURRENTRW=NO means that you want the system to log all errors that occur due to reads that are concurrent with writes.

If you code CONCURRENTRW=YES, then EXTLOCK is the default. EXTLOCK means that your program expects serialization on a data set extent basis. This is the serialization that is normally provided by a device that can be read or written. This applies to any type of data set.

CONCURRENTRW=(YES,TRKLOCK) - Specify this option if your program might access a data set on a secondary device of a PPRC pair when the device is defined with the read-only attribute. This device bypasses extent collision checking. An application should specify TRKLOCK only if it can tolerate inconsistent data due to concurrent writes to other tracks. This option has an effect only when the device is defined as read-only.

Source: You might set this parameter in the DCBE macro or in the DCB OPEN exit routine. It should not be changed while the DCB is open.

DSENCRYPT={OK|NOTOK}

Specifies the support for EXCP access of encrypted basic and large format data sets. (Only applies to basic and large format data sets. EXCP access is not supported for extended format data sets or PDSEs.)

DSENCRYPT=OK allows you to specify that your EXCP application program supports how to process encrypted data sets according to BSAM and QSAM requirements. The specification of this resolves to the DCBEDSENCRYPTOK indicator in the DCBE to be set on. This option is needed when processing an encrypted data set in either of these situations:

- The DCB indicates MACRF=E (the DCB is only for EXCP). OPEN tests this new bit. If you do not code this option, the OPEN function issues an ABEND 213-9A.
- The DCB is for BSAM or QSAM, but the application program issues EXCP or EXCPVR. The system tests this bit for each EXCP or EXCPVR. The user program must also set on the IOBSPSVC bit in the IOB. It informs the access method not to examine this IOB. For more information, see IOBSPSVC in *z/OS DFSMSdfp Advanced Services*. If you do not code this option, the EXCP or EXCPVR fail.

Note that the DCBEDSENCRYPTOK indicator means that the system acknowledges that you have specified DSENCRYPT=OK. It does not mean that the data set is encrypted or that the application program issues EXCP, EXCPVR, or XDAP.

DSENCRYPT=NOTOK indicates that the EXCP application does not support encrypted data sets. The specification of this resolves to the DCBEDSENCRYPTOK indicator in the DCBE to be set off. This is the default.

Note: DCBEDSENCRYPTOK is checked during:

- open with EXCP DCB
- I/O time for EXCP callers

EADSCB=OK|NOTOK

Specify the support level for extended attribute DSCBs.

EADSCB=OK allows you to specify that your application program supports the following:

- Opening a VTOC that might have format 8 DSCBs. These calls must provide a DCBE macro with the EADSCB=OK keyword to indicate that the caller supports extended attribute data provided in DSCBs and track addresses with cylinder 65520 or larger. If you do not code this option, the OPEN function issues ABEND 113-48 and message IEC142I. Code this option when your application program supports Format 8 and 9 DSCBs.
- Opening a data set that has a format 8 DSCB for EXCP access or for BDAM access with OPTCD=A. These calls must provide a DCBE macro with the EADSCB=OK keyword to indicate that the caller supports extended attribute data provided in DSCBs and track addresses with cylinder 65520 or larger. If you do not code this option, the OPEN function issues ABEND 113-44 and message IEC142I. Code this option when your application program supports Format 8 and 9 DSCBs and such track addresses.

EADSCB=NOTOK indicates a calling program does not support extended attribute DSCBs. The specification of this resolves to the DCBEEADSCBOK indicator in the DCBE to be set off. This is the default.

EODAD=*relexp*
specifies the address of an end-of-data routine given control when the end of an input data set is reached. The entry point might be above the line or below the line. If the EODAD routine resides above the line, you must issue all CHECKs or GETs in 31-bit addressing mode.

An EODAD address in the DCBE takes precedence over an EODAD address in the DCB. The EODAD routine (whether it is specified in the DCBE or DCB) gets control in the addressing mode in which the CHECK or GET is issued.

If the record format is RECFM=FS or FBS, the end-of-data condition is detected when a file mark is read or when more data is requested after reading a truncated block.

If the end of data block is reached but no EODAD address was supplied in either the DCBE or DCB, or if a GET macro is issued after an end-of-data exit is taken, the task is abnormally terminated. For more information about the EODAD routine, see *z/OS DFSMS Using Data Sets*. You might also see the EODAD parameter in the appropriate DCB macro.

Source: EODAD can be supplied in the DCBE macro or by the problem program before the end of the data set is reached.

FIXED=USER

With this DCBE option, you assert that the data areas remain fixed from the time the READ or WRITE macro instruction is issued through the completion of the CHECK or WAIT macro instruction. Failure to keep the data areas fixed results in a system integrity exposure as the channel program uses the real addresses associated with the data areas.

The purpose of this option is to improve performance by using less processor time.

Your program can ensure that the data areas are fixed by doing one of the following:

- Issuing the PGSER FIX macro
- Using the GETMAIN or STORAGE macro for a page fixed subpool
- Issuing the IARV64 macro with REQUEST=GETSTOR and TYPE=DREF
- Issuing the IARV64 macro with REQUEST=PAGEFIX
- Issuing the IARST64 macro with REQUEST=GET and TYPE=DREF or TYPE=FIXED
- Issuing the IARCP64 macro with REQUEST=BUILD and TYPE=DREF or TYPE=FIXED.

All of these methods of fixing pages require that your program has a form of authorization, such as APF authorization or running in either supervisor state or system protection key. Other restrictions might apply. The FIXED=USER option also requires one of these forms of authorization.

This parameter has an effect only for these types of DASD data sets:

- Basic or large format
- Extended format but not compressed format
- Partitioned but not PDSE.

If the data set is not one of those types, the system takes care of any page fixing that is needed.

The following is not intended programming interface information: After completion of the OPEN macro, your program can test a bit to determine whether FIXED=USER has an effect. The bit is DEB2XUPF in the DEB2XFG3 byte and is mapped by the IEZDEB macro in DEB2X.

GETSIZE={YES|NO}
specifies that OPEN is to calculate the number of blocks in the data set and store this number in the DCBE (DCBESIZE). In most cases this is an estimate. With concatenated data sets the number is for only the current data set. As you read through the data sets, the system changes this number.

DCBESIZE is valid after OPEN and on entry to the user's DCB OPEN exit routine. However, for compressed format data sets, DCBESIZE is not valid until after OPEN.

For a compressed format data set, the number of physical blocks in the data set differs from the number of user blocks that are found in the data set. DCBESIZE refers to the number of user blocks found in the data set.

This parameter is ignored if the data set is not extended format data sets or UNIX files.

For UNIX files,

- If GETSIZE=YES is specified, DCBEXSIZ (an 8-byte value) is set to the approximate number of blocks in the file based on DCBRECFM and DCBBLKSI.

  GETSIZE is not supported for FIFO or character special files. DCBEXSIZ is set to 0.

Source: You might set this parameter in the DCBE macro.

HYPERWRITE={YES|NO}
specifies the user's request for zHyperWrite, which can result in better performance of writes in a HyperSwap environment. The caller must be APF authorized, running in supervisor state or system key. If the associated data set is not an extended format data set opened for OUTPUT, INOUT, or OUTIN, this parameter is ignored.

IBM zHyperWrite processing can be used with write I/O operations to perform software mirroring to peer-to-peer remote copy (PPRC) devices that are monitored for HyperSwap processing (with GDPS or IBM Copy Services Manager). IBM zHyperWrite data replication can be used to reduce latency in these HyperSwap environments. Maximum benefit is realized when IBM zHyperWrite data replication is used and all synchronously mirrored relationships are managed by HyperSwap.

LOC={ANY|BELOW}
Specify this option before issuing the OPEN or RDJFCB macro. If you specify LOC=BELOW or do not code LOC=, you signify that the program does not support the XTIOT, UCB NOCAPTURE, or DSAB-above-the-line options of dynamic allocation. These are the S99TIOEX, S99ACUCB, and S99DSABA options. By setting LOC=ANY you signify that the program is either not affected by or that is allowed for any of the following possibilities:

- The DCBTIOT field (offset in TIOT to an entry) might contain zeros or contain a TIOT offset.
- The DEBXDSAB field (address of DSAB) might point above the line.
- The DSABTIOT field might point to an XTIOT or to a TIOT entry.
- The UCB address field in the DEB might be four bytes or three bytes (test the DEB31UCB bit).
- The TIOEFSRT field might contain zeros instead of a UCB address.

For more information on The XTIOT and other dynamic allocation options, see *z/OS DFSMS Using Data Sets* and *z/OS MVS Programming: Authorized Assembler Services Guide*.

Regardless of dynamic allocation and the NON_VSAM_XTIOT setting, you should always specified DCBE LOC=ANY when either your program does not reference TIOT, UCB, and DSAB, or that it is correctly modified to support the XTIOT, UCB NOCAPTURE, or DSAB-above-the-line options.

Table 29. LOC=ANY for BSAM, QSAM, and BPAM

NON_VSAM_XTIOT=	DCBE LOC=	Result
NO or not coded	BELOW or not coded	OPEN return code 8, Message IEC133I, DCBOFOPN bit is off.
NO or not coded	ANY	ABEND 113–4C, messages IEC133I and IEC142I.
YES	BELOW or not coded	OPEN return code 8, Message IEC133I, DCBOFOPN bit is off.
YES	ANY	Successful OPEN.

If the application program specifies LOC=ANY for a data set dynamically allocated with the options XTIOT, UCB NOCAPTURE, and/or DSAB-above-the-line, and the NON_VSAM_XTIOT=YES option in

PARMLIB is in effect, then OPEN sets the two-byte DCBTIOT field to zero instead of setting it to an offset in the TIOT, and the data set is opened successfully. However, if the application program sets the DCBE option LOC=ANY before OPEN, but the NON_VSAM_XTIOT=YES option in PARMLIB is not in effect and any of the three dynamic allocation options is in effect, then OPEN issues an ABEND 113–4C and message IEC142I. A summary of the expected results with BSAM, QSAM, and BPAM is in Table 29 on page 236.

Table 30. LOC=ANY for EXCP

NON_VSAM_XTIOT=	DCBE LOC=	Result
NO or not coded	BELOW or not coded	OPEN captures the UCB if needed and CLOSE uncaptures it.
NO or not coded	ANY	OPEN captures the UCB if needed and CLOSE uncaptures it. OPEN issues message IEC136I ddname, DCBE LOC=ANY NOT HONORED DUE TO PARMLIB OPTION.
YES	BELOW or not coded	OPEN captures the UCB if needed and CLOSE uncaptures it.
YES	ANY	Successful OPEN and UCB are not captured.

The LOC=ANY option for an EXCP DCB for DASD or tape means that the application program accepts:

- OPEN not capturing the UCB,
- The TIOT DD entry being an XTIOT, and
- The DSAB being above the line.

The result depends on the NON_VSAM_XTIOT option of the DEVSUPxx member of PARMLIB as described in Table 30 on page 237. An EXCP DCB for a device other than DASD or tape continues to get the existing failures.

MULTACC=*n*

allows the system to process BSAM I/O requests more efficiently by not starting I/O until a number of buffers have been presented to BSAM.

A nonzero value indicates to OPEN that BSAM can do a more efficient type of queuing of (accumulation) of READ or WRITE requests. If you code a nonzero value, your program must not issue a WAIT or EVENTS macro against a DECB unless you preceded it with issuance of a TRUNC macro. If you code a nonzero value but your program issues a WAIT or EVENTS macro against a DECB for the DCB and the program did not issue a TRUNC after the previous READ or WRITE, the program might go into an unending wait.

If your program follows the rules for MULTACC use but the data set type does not support it, the program runs correctly.

If you code a nonzero value, OPEN calculates a default number of READ or WRITE requests that you are suggesting the system queue more efficiently. First OPEN calculates the number of BLKSIZE-length blocks that can fit on a track. OPEN then multiplies this value by the MULTACC value and for an extended format data set, by the number of stripes. The system tries to defer starting I/O requests until you have issued this many READ or WRITE requests for the DCB. BSAM never queues (defer) more READ or WRITE requests than the NCP value set in OPEN.

MULTACC has an effect only for BSAM DASD nonspooled, and non-PDSE data sets. In the current release, it has no effect for other types of data sets or UNIX files.

MULTACC has no effect for compressed format data sets. The user might issue WAITs in this case.

Recommendation: IBM recommends that users not take advantage of this characteristic of compressed format data sets (that WAIT might be issued although MULTACC is specified) because it does not work reliably for other types of data sets and in future levels of the system, it might not work with compressed format data sets.

Source: You might set MULTACC in the DCBE macro or in the DCB OPEN exit routine. This parameter should not be changed while the DCB is open except when the DCB OPEN exit is reentered for each data set in a concatenation where you have set on the DCB unlike attributes bit.

MULTSDN=*n*
requests a system-defaulted NCP.

If nonzero and DCBNCP are zero and the data set block size is available, the system calculates an appropriate initial NCP value. The system then multiplies this value by the number specified in MULTSDN and store this value in DCBNCP. DCBNCP is set before the DCB OPEN exit routine is given control. This allows you to give the system indicators without being dependent on device information such as blocks per track or number of stripes. If DCBNCP is zero after returning from the OPEN exit, the SDN is derived or rederived after the OPEN exit and stored in DCBNCP.

If you are using large block interface (LBI) tape, DCBNCP is set to MULTSDN with the value being a minimum of 2 and a maximum of 16. For non-LBI tape, the default DCBNCP is 5.

For DASD data sets which are not extended format data sets, the initial NCP value is the number of DCBBLKSI-length blocks that can fit on a track.

For extended format data sets (not in the compressed format), the initial NCP value is the number of DCBBLKSI-length blocks (plus the suffix) that can fit on a track times the number of stripes. For compressed format data sets, the initial NCP value is 1 because 1 is the most efficient value.

For UNIX files, if MULTSDN is specified (and DCBNCP is not specified), DCBNCP is set to the value specified by MULTSDN. Currently, the initial NCP value is set to 5.

Restriction: This parameter is ignored if BLKSIZE is not available from any source.

The NCP limit is 255.

Source: You might set MULTSDN in the DCBE macro or in the DCB OPEN exit routine. This parameter should not be changed while the DCB is open except when the DCB OPEN exit is reenterederivedd for each data set in a concatenation where you have set on the DCB unlike attributes bit.

NOVER={YES|NO}
specifies that OPEN should bypass any verification to determine whether the size of the stripes of an extended format data set are consistent. The default is NO.

Inconsistent stripes might be caused by inadvertently restoring one or more stripes of an extended format data set without restoring all stripes.

In general, OPEN uses the longest stripe to be the end of the file if you specify NOVER=YES. However, if the longest stripe fills a track and a later stripe ends in the middle of that same relative track, OPEN assumes the shorter stripe to be the true data set end.

This parameter is ignored if th e data set is not an extended format data set.

Source: You might set this parameter in the DCBE macro or in the DCB OPEN exit routine. It should not be changed while the DCB is open.

PASTEOD={YES|NO}
specifies that the end-of-data marker of the extended format data set, which is saved when the data set is open for INPUT, UPDATE, OUTIN, or INOUT is to be ignored. The default is NO.

This parameter is ignored if the data set is not an extended format data set. This parameter is ignored if the data set is open for other than INPUT, INOUT, UPDAT, or OUTIN.

For extended format data sets, the system saves the end-of-data marker of the data set when the data set is opened for input or update. If the data set is opened for output while it is still open for input (without specifying PASTEOD=YES), the input DCB does not see any of the records which might be written past the end-of-data marker by the output DCB. PASTEOD=YES allows the input DCB to read past the end-of-data marker of the data set which was saved when the data set was opened. This allows the input DCB to read records which might have been written past the end-of-data marker by another DCB.

Source: You might set this parameter in the DCBE macro or in the DCB OPEN exit routine. It should not be changed while the DCB is open.

RMODE31={BUFF|NONE}

specifies whether you request that OPEN get QSAM buffers above the 16 MB line (RMODE31=BUFF) or not (RMODE31=NONE) when acquiring buffers automatically. The default is NONE. If BFTEK=A is specified in the DCB, OPEN also gets the QSAM logical record interface (LRI) area above the 16MB line. CLOSE frees these buffers and the LRI area, if it exists.

In releases prior to DFSMS/MVS 1.1, FREEPOOL is typically issued after CLOSE since CLOSE does not free the QSAM 24-bit buffers. However, if OPEN honors your request for buffers above the 16 MB line, you should either avoid the FREEPOOL macro, or reassemble the program with the FREEPOOL macro. At DFSMS/MVS 1.1, the FREEPOOL expansion tests whether the buffer pool exists before attempting to free it.

The RMODE31=BUFF parameter has no effect if any of the following are true:

- BUFCB is specified on the DCB macro.
- Buffer pool is built by a GETPOOL, BUILD, or BUILDRCD macro or a previous OPEN.
- Access method is BSAM or BPAM.
- OPEN leaves DCBEMD31 as zero.

Source: You might set this parameter in the DCBE macro or in the DCB OPEN exit routine. It should not be changed while the DCB is open except when the DCB OPEN exit is reentered for each data set in a concatenation where you have set on the DCB unlike attributes bit.

SYNAD=*relexp*

specifies the address of an error analysis (SYNAD) routine given control when an uncorrectable input/output error occurs. The entry point might be above the line or below the line. If the SYNAD routine resides above the line, you must issue all CHECKs, GETs, or PUTs in 31-bit addressing mode.

A SYNAD address in the DCBE takes precedence over a SYNAD address in the DCB. The SYNAD routine (whether it is specified in the DCBE or DCB) gets control in the addressing mode in which the CHECK, GET, or PUT is issued.

If an uncorrectable input/output error is encountered but no SYNAD routine is supplied in either the DCBE or DCB, the task is abnormally terminated. See *z/OS DFSMS Using Data Sets* for more information about the SYNAD routine. You might also see the SYNAD parameter in the appropriate DCB macro.

Source: SYNAD can be supplied in the DCBE macro or by the problem program. The problem program can also change the error routine address at any time.

SYNC={SYSTEM|NONE|(NUMFILES,*nnn*)}

This parameter is intended for use with magnetic tapes. These options are available: SYSTEM, NONE and (NUMFILES,*nnn*). The default is SYSTEM.

You can use the NUMFILES keyword to specify the number of files that a job can submit sequentially before they are written to tape (synchronized). For example, if you specify a value of NUMFILES(100), then up to 100 files can be written specifying PASS RETAIN or CLOSE LEAVE before the files are explicitly written to the tape medium.

If you specify SYSTEM (the default value), then when your program is in write mode, the system tries to ensure that your data is safe on the medium in each of the following circumstances:

- The program switches to another volume to continue writing.
- The data set is closed.

If the system's volume-switching function or close function detects a data loss, it issues an ABEND.

Issuing a BSAM CHECK macro or EXCP WAIT macro ensures that the data has been sent to the device. Whether it is safe on the medium depends on the type of device and data set, and on the guaranteed synchronous write option in the storage class. This storage class option affects only PDSEs. You can use the SYNCDEV macro to ensure synchronization, but the performance might not be up to your installation's standards.

VERSION={0|1}

Specifies the version of the DCBE to be generated.

Version 0 results in a DCBE expansion of 56 bytes. Version 1 results in a DCBE expansion of 80 bytes.

The default is 0, unless BYPASS_AUTH is specified. Explicitly coding VERSION=0 is not valid when BYPASS_AUTH= is specified.

QSAM support for MULTSDN

QSAM uses the MULTSDN value in the DCBE macro to calculate a more efficient BUFNO value for tape data sets and specific types of DASD data sets, and reduces the situations where you need to specify a BUFNO value. For concatenated data sets, QSAM uses MULTSDN to dynamically recalculate the BUFNO value when switching from one concatenated data set to the next.

QSAM accepts a MULTSDN value for the following data sets:

- tape data sets
- DASD data sets of the following types:
  - basic format
  - large format
  - extended format (non-compressed)
  - PDS.

For the supported types of data sets, the system uses MULTSDN to calculate a more efficient value for BUFNO when the following conditions are true:

- The MULTSDN value is not zero.
- DCBBUFNO has a value of zero after completion of the DCB OPEN exit routine
- The data set block size is available.

When MULTSDN is specified, note that the default number of buffers may be less than what would have been derived without MULTSDN, as shown in Table 31 on page 240.

Table 31. Default buffer numbers for QSAM with and without MULTSDN

Data Set Type	DCBBUFNO default without MULTSDN	DCBBUFNO default with MULTSDN
PDSE Member	1	1
Extended format data set in the compressed format	1	1
UNIX file	1	1
Extended format data set (not in the compressed format)	2 * number of stripes * number of blocks per track	MULTSDN * number of stripes * number of blocks per track
Block size equal to or greater than 32 KB (tape)	2	MULTSDN value

Table 31. Default buffer numbers for QSAM with and without MULTSDN (continued)		
Data Set Type	**DCBBUFNO default without MULTSDN**	**DCBBUFNO default with MULTSDN**
Block size less than 32 KB (tape)	5	MULTSDN * number of blocks in 64 KB
IBM 2540 card reader or card punch	3	3
PS, PDS	5	MULTSDN * number of blocks per track
Others, including dummy data sets	5	5
TSO terminal	5	1

For more information, see the description of the MULTSDN parameter of the DCBE macro in *z/OS DFSMS Macro Instructions for Data Sets*.

BSAM and QSAM support for MULTACC on tape

In z/OS V1R9, the MULTACC parameter of the DCBE macro is expanded, to optimize performance for tape data sets with BSAM, and to support QSAM with optimized performance for both tape and DASD data sets. The calculations used to optimize performance for BSAM with DASD data sets are also enhanced.

For BSAM in V1R9, if you code a nonzero MULTACC value, OPEN calculates a default number of READ or WRITE requests that you are suggesting the system queue more efficiently. OPEN calculates the number of BLKSIZE-length blocks that can fit within 64 KB, then multiplies that value by the MULTACC value. If the block size exceeds 32 KB, then OPEN uses the MULTACC value without modification (this can happen only if you are using LBI, the large block interface). The system then tries to defer starting I/O requests until you have issued this number of READ or WRITE requests for the DCB. BSAM will never queue (defer) more READ or WRITE requests than the NCP value set in OPEN.

For QSAM in V1R9, if you code a nonzero MULTACC value, OPEN calculates a default number of buffers that you are suggesting the system queue more efficiently. OPEN calculates the number of BLKSIZE-length blocks that can fit within 64 KB, then multiplies that value by the MULTACC value. If the block size exceeds 32 KB, then OPEN uses the MULTACC value without modification (this can happen only if you are using LBI, the large block interface). The system then tries to defer starting I/O requests until that number of buffers has been accumulated for the DCB. QSAM will never queue (defer) more buffers than the BUFNO value that is in effect.

IBM recommends setting MULTACC to one half of the MULTSDN value.

If you code a MULTACC value that is too large for the system to use, the system ignores the excess amount. However, the absolute upper limit for MULTACC is 255.

For more information, see the description of the MULTACC parameter of the DCBE macro in *z/OS DFSMS Macro Instructions for Data Sets*.

Buffered tape marks

Two options are available: SYSTEM and NONE. SYSTEM is the default.

System

When your program is in write mode, the system will try to ensure that your data is safe on the medium in each of the following circumstances:

- The program switches to another volume to continue writing.
- The data set is closed.

If the system's volume-switching function or close function detects a data loss, it issues an ABEND.

Issuing a BSAM CHECK macro or EXCP WAIT macro ensures that the data has been sent to the device. Whether it is safe on the medium depends on the type of device and data set, and on the guaranteed synchronous write option in the storage class. This storage class option affects only PDSEs. You can use the SYNCDEV macro to ensure synchronization, but the performance might not be up to your installation's standards.

NONE

If the device supports buffered tape marks, the OPEN, EOV, and CLOSE functions take advantage of it when writing. This can save several seconds of real time. If the device does not support buffered tape marks, this option has no effect. Similarly this option has no effect on an older level of the system. SYNC=NONE might have an effect on other device types. This option does not affect the final tape mark written by the EOV function.

Specifying the option NONE can affect data integrity. If this option is in effect when making the transition to a new volume or when closing the data set, the system does not ensure that user data, tape marks, and data set labels are safely on the medium.

Use this option only under the following conditions:

- Your data is not important. For example, if it is test data, or you are measuring performance.
- Your program turns the option on before opening the DCB and turns the option off after the OPEN. This allows the OPEN to be much faster. If there is an I/O error while the data set labels or tape mark are being written, OPEN might not detect it, but it is reflected either while your program is writing or during CLOSE. If it is reflected while your program is writing, the system calls your program's SYNAD routine and if it is not available, the system issues an ABEND 001. If your data set is relatively small and the system does not detect the header label I/O error until CLOSE, CLOSE issues an ABEND if this option is still in effect. Your program can turn this option on or off at any time.
- Your program is writing multiple files on the tape and your program has a method of recovering the loss of multiple files. For example, your program might be designed to write multiple files and is not successful unless all the files are written. You should turn SYNC=NONE off before the last CLOSE or before the last OPEN.

DESERV—Directory entry services (BPAM)

The DESERV macro performs operations on PDS and PDSE directories.

The DESERV FUNC= parameter specifies the function requested, such as:

- GET—retrieve directory information for a PDS or PDSE given a list of names or BLDL entry
- GET_ALL—retrieve all member names and directory entries from a PDSE or PDS
- RELEASE—removes PDSE connections established by previous DESERV functions such as GET and GET_ALL
- GET NAMES—gets a list of names and associated directory data for a member of a PDSE
- RENAME PDSE members and alias names
- DELETE PDSE entries
- UPDATE selected fields of program object directory entries
- GET_G— retrieves all the generation information for a single member of a PDSE
- GET__ALL_G— retrieves all the generation information for all the members of a PDSE.

DESERV returns directory information in system managed directory entry (SMDE) format depending upon the type of request you specify. The SMDE contains reformatted PDS2 information for a PDS member plus additional information for a PDSE program object. See *z/OS DFSMS Using Data Sets* for more information on using the DESERV functions.

The DESERV macro may be issued in 24- or 31-bit addressing mode. In either case, all addresses must be valid 31-bit addresses.

The DESERV exit and the PUT function are not documented here. See *z/OS DFSMSdfp Advanced Services* for more information on these functions.

The syntax for each DESERV function is shown below. Table 32 on page 246 and Table 33 on page 247 show the parameters which are either required, optional, or invalid for each of the DESERV functions. The parameter descriptions follow the figures. The return and reason codes for each DESERV function are shown in the figures that follow the parameters.

The parameter list is cleared for the execute form of DESERV (MF=E) if the COMPLETE parameter is specified. This can be used to reset previously used parameters. You are responsible for initializing the parameter list by copying MF=L to dynamic storage for use in MF=E.

DESERV—Function=DELETE

DESERV FUNC=DELETE deletes member names and aliases from a PDSE directory. When a member name is deleted, all alias names are automatically deleted.

The format of the DESERV FUNC=DELETE macro is:

[*label*]	DESERV	FUNC=DELETE ,DCB=*data_control_block* ,NAME_LIST=(*input_list*,*input_list_entry_count*) [,MF={(E,*parmlist_name*[,NOCHECK|COMPLETE])|S}] [,RETCODE=*return_code*] [,RSNCODE=*reason_code*]

DESERV—Function=GET

DESERV FUNC=GET retrieves directory entry information for a list of names or a BLDL directory entry that you provide. The directory entries are returned in system managed directory entry (SMDE) format.

The format of the DESERV FUNC=GET macro is:

[*label*]	DESERV	FUNC=GET {,AREA=(*buffer_area*, *buffer_area_size*) | ,AREAPTR=*buffer_area_address}* [,SUBPOOL=*subpool_id*] [,BYPASS_LLA={YES|NO}] [,EXT_ATTR={YES|NO}] [,CONN_ID=*connection_addr*] ,CONN_INTENT=HOLD ,DCB=*data_control_block* [,ENTRY_GAP={*gap_size*|0}] {,NAME_LIST=(*input_list*,*input_list_entry_count)*| PDSDE=*BLDL_directory_entry}* [,MF={(E,*parmlist_name*[,NOCHECK|COMPLETE])| S}] [,RETCODE=*return_code*] [,RSNCODE=*reason_code*]

DESERV—Function=GET_ALL

DESERV FUNC=GET_ALL retrieves SMDEs for all member names (primary and alias) of a PDSE and can establish connections to members.

The format of the DESERV FUNC=GET_ALL macro is:

[*label*]	DESERV	FUNC=GET_ALL ,AREAPTR=*buffer_area_address* [,CONCAT={*concat_number*|ALL}] [,CONN_ID=*connection_addr*] [,CONN_INTENT={NONE|HOLD}] ,DCB=*data_control_block* [,EXT_ATTR={*gap_size*|0}] [,HIDE={YES|NO}] [,HIDE={YES|NO}] [,MF={(E,*parmlist_name*[,NOCHECK|COMPLETE])| S}] [,RETCODE=*return_code*] [,RSNCODE=*reason_code*] [,SUBPOOL=*subpool_id*]

DESERV—Function=GET_ALL_G

DESERV FUNC=GET_ALL_G retrieves directory entry information for the members of a PDSE. When a generation name is provided it will start with the next generation after the generation passed. The directory entries are returned in system managed directory entry (SMDE) format with a SMDE_GENE extension. A generationname is a field composed of an 8-byte member name, followed by a 4-byte absolute generation number.

GET_ALL_G works only for non-concatenated data sets.

The format of the DESERV FUNC=GET_ALL_G macro is:

[*label*]	DESERV	FUNC=GET_ALL_G ,AREA=(*buffer_area*,*buffer_area_size*) ,DCB=*data_control_block* [,NAME_LIST=(*generationname*,1)] [,MF={(E,*parmlist_name*[,NOCHECK|COMPLETE])|S}] [,RETCODE=*return_code*] [,RSNCODE=*reason_code*]

DESERV—Function=GET_G

DESERV FUNC=GET_G retrieves generation data for a member of a PDSE that you provide. The directory entries are returned in system managed directory entry (SMDE) format with an SMDE_GENE extension defined in IGWSMDE. A *generationname* is a field composed of an 8-byte member name, followed by a 4-byte absolute generation number. An 8-byte member name should be passed on the first call. If there is not sufficient space to return all the data, the last generation return should be passed on subsequent calls.

GET_G works only for non-concatenated data sets.

The format of the DESERV FUNC=GET_G macro is:

[*label*]	DESERV	FUNC=GET_G ,AREA=(*buffer_area*,*buffer_area_size*) ,DCB=*data_control_block* ,NAME_LIST=(*generationname*,1) [,MF={(E,*parmlist_name*[,NOCHECK|COMPLETE])|S}] [,RETCODE=*return_code*] [,RSNCODE=*reason_code*]

DESERV—Function=GET_NAMES

DESERV FUNC=GET_NAMES, obtains a list of all names and associated data for a member of a PDSE.

The format of the DESERV FUNC=GET_NAMES macro is:

[*label*]	DESERV	FUNC=GET_NAMES ,AREAPTR=*buffer_area_address* [,CONCAT=*concat_number*] ,DCB=*data_control_block* ,NAME=*name_record* [,MF={(E,*parmlist_name*[,NOCHECK|COMPLETE])| ___S}] [,RETCODE=*return_code*] [,RSNCODE=*reason_code*] [,SUBPOOL=*subpool_id*]

DESERV—Function=RELEASE

DESERV FUNC=RELEASE removes connections established by GET or GET_ALL functions.

The format of the DESERV FUNC=RELEASE macro is:

[*label*]	DESERV	FUNC=RELEASE {CONN_ID=*connection_addr*\ DE_LIST=(*input_list*,*input_list_entry_count*)} ,DCB=*data_control_block* [MF={(E,*parmlist_name*[,NOCHECK|COMPLETE])| ___S}] [,RETCODE=*return_code*] [,RSNCODE=*reason_code*]

DESERV—Function=RENAME

DESERV FUNC=RENAME renames member and alias names in a PDSE.

The format of the DESERV FUNC=RENAME macro is:

[*label*]	DESERV	FUNC=RENAME ,DCB=*data_control_block* ,NAME_LIST=(*input_list*,*input_list_entry_count*) [,MF={(E,*parmlist_name*[,NOCHECK|COMPLETE])| ___S}] [,RETCODE=*return_code*] [,RSNCODE=*reason_code*]

DESERV—Function=UPDATE

DESERV FUNC=UPDATE allows you to update selected attributes of program objects in a PDSE. See the DESERV UPDATE function in *z/OS DFSMS Using Data Sets* for more information on the fields which can be updated.

The format of the DESERV FUNC=UPDATE macro is:

[*label*]	DESERV	FUNC=UPDATE ,DCB=*data_control_block* ,NAME_LIST=(*input_list*,*input_list_entry_count*) [,MF={(E,*parmlist_name*[,NOCHECK|COMPLETE])| S}] [,RETCODE=*return_code*] [,RSNCODE=*reason_code*]

DESERV—List form

DESERV MF=L is the list form of the DESERV macro.

The format of the DESERV MF=L macro is:

[*label*]	DESERV	[parms...] ,MF=L

Table 32 on page 246 and Table 33 on page 247 show the DESERV macro parameters and indicate for each function if the parameter is required, optional, or invalid. The figure applies to the MF=S (standard) forms of the macro, or to the logically merged MF=L and MF=E parameters.

Table 32. DESERV keyword parameters by function

Keyword / FUNC=	GET	GET_ALL	RELEASE	GET_NAMES
AREA	Optional	Invalid	Invalid	Invalid
AREAPTR	Optional	Required	Invalid	Required
BYPASS_LLA	Optional	Invalid	Invalid	Invalid
CONCAT	Invalid	Optional	Invalid	Optional
CONN_ID	Optional	Optional	Optional	Invalid
CONN_INTENT	Required	Optional	Invalid	Invalid
DCB	Required	Required	Required	Required
DE_LIST	Invalid	Invalid	Optional	Invalid
ENTRY_GAP	Optional	Optional	Invalid	Invalid
FUNC	Required	Required	Required	Required
HIDE	Invalid	Optional	Invalid	Invalid
MF	Optional	Optional	Optional	Optional
NAME	Invalid	Invalid	Invalid	Required
NAME_LIST	Optional	Invalid	Invalid	Invalid
PDSDE	Optional	Invalid	Invalid	Invalid
RETCODE	Optional	Optional	Optional	Optional
RSNCODE	Optional	Optional	Optional	Optional
SUBPOOL	Optional	Optional	Invalid	Optional

Table 33. DESERV keyword parameters by function

Keyword / FUNC=	UPDATE	RENAME	DELETE
AREA	Invalid	Invalid	Invalid
AREAPTR	Invalid	Invalid	Invalid
BYPASS_LLA	Invalid	Invalid	Invalid
CONCAT	Invalid	Invalid	Invalid
CONN_ID	Invalid	Invalid	Invalid
CONN_INTENT	Invalid	Invalid	Invalid
DCB	Required	Required	Required
DE_LIST	Invalid	Invalid	Invalid
ENTRY_GAP	Invalid	Invalid	Invalid
FUNC	Required	Required	Required
HIDE	Invalid	Invalid	Invalid
MF	Optional	Optional	Optional
NAME	Invalid	Invalid	Invalid
NAME_LIST	Required	Required	Required
PDSDE	Invalid	Invalid	Invalid
RETCODE	Optional	Optional	Optional
RSNCODE	Optional	Optional	Optional
SUBPOOL	Invalid	Invalid	Invalid

AREA=(*buffer_area*,*buffer_area_size*) *buffer_area*—MF=S form, RX–type address or (2-12) *buffer_area*—MF=L form, A–type address) *buffer_area*—MF=E form, RX–type Address or (2-12)

Specifies an area provided by the caller into which the GET function stores directory entries.

The area is mapped by the DESB DSECT in the IGWDES mapping macro on return from the function. The storage must be modifiable in the key of the caller.

AREA and AREAPTR are mutually exclusive.

If the area is filled before the processing has ended, the request is terminated at that point. The entries in the buffers are valid and connections may have been established.

buffer_area_size—Symbol or (2-12)
absexp or (2-12)—Standard or execute form
absexp—List form

buffer_area_size is the length in bytes of the area specified in the *buffer_area* parameter.

Restriction: There is no way to determine, in advance, the exact buffer size required to contain the directory entries on a single request. A formula for length calculation is provided in Figure 5 on page 248.

FORMULA:
buffer_area_size = L'DESB_FIXED + (*input_list_entry_count* * (SMDE_MAXLEN + *gap_size*))

WHERE:

buffer_area_size
is the storage required to hold *input_list_entry_count* number of entries.

DESB_FIXED
is the fixed (header) portion of the buffer. It is a constant defined by the IGWDES macro.

SMDE_MAXLEN
is a constant defined by macro IGWSMDE that defines the current maximum size of a single SMDE entry. This is a very large value because names can be up to 1024 bytes in length.

gap_size
is the value specified by the ENTRY_GAP parameter on the GET or GET_ALL function.

input_list_entry_count
is the value passed on the NAME_LIST parameter for the number of entries in the list or 1 if PDSDE is specified.

Figure 5. Buffer size calculation for GET function.

AREAPTR=*buffer_area_address* A–type address or (2-12). Standard form RX–type address or (2-12). Execute form A–type address. List form
specifies a word where GET, GET_ALL, and GET_NAMES store the address of the first DESB buffer output.

The buffer-area address points to a chain of buffers mapped by the DESB mapping in the IGWDES mapping macro on return from the function.

The subpool number for the storage obtained is placed in the buffer header. See the description of the SUBPOOL keyword for subpool value determination.

It is your responsibility to release the storage using the STORAGE or FREEMAIN macro.

If you issue a DESERV call while running in 24-bit addressing mode, the storage area returned will be below the 16 MB line. If you issue a DESERV call while running in 31-bit addressing mode, the storage returned can be above or below the 16 MB line.

AREAPTR and AREA are mutually exclusive.

BYPASS_LLA={YES | NO}
indicates whether the GET function should bypass LLA's cached directory entries and go only to the current library directory or use LLA's cached directory entries if they are available.

BYPASS_LLA=YES indicates that the LLA cache is not examined. BYPASS_LLA=NO, the default, indicates that the LLA cache is examined before attempting to obtain information directly from the data set.

This is an optional parameter to the GET function.

Currently, the GET_ALL function does not obtain member list from LLA. Therefore, the directory entries come directly from the data set as though BYPASS_LLA=YES were specified.

Tip: Response time is better if the directory entries are obtained from LLA.

CONCAT={*concat_number*|ALL}
specifies the library concatenation.

***concat_number Absexp or (2-12)* Standard and execute form. *Absexp* List form.**
specifies for the GET_ALL and GET_NAMES function the specific library in a concatenation of libraries. DESERV returns all the member names. *concat_number* is a numeric value in the range of 0 to 255.

This is an optional parameter and the default is the first library in the concatenation (that is, 0).

ALL
specifies (for the GET_ALL function only) to return all the names in each PDS or PDSE directory in the concatenation. DESERV returns a list of directory entries which contains a merged list of SMDEs from each data set in the concatenation where duplicate member names have been eliminated.

CONN_ID=*connection_addr* A—type address or (2-12). Standard form RX—type address or (2-12). Execute form A—type address. List form
specifies the location of the 4-byte value used by the GET, GET_ALL, and RELEASE functions. The four bytes are a token that relates connections to a particular invocation of a function. It may indicate a number of connections or no connections at all.

For the RELEASE function, CONN_ID is an input parameter and is mutually exclusive with DE_LIST.

For the GET and GET_ALL functions, CONN_ID is an output parameter.

CONN_ID is meaningful only when one or more of the designated libraries are PDSEs.

Note: A maximum of 65536 connection identifiers per DCB can exist simultaneously. You can free the identifier by using the RELEASE function and specifying the connection identifier to be freed. The identifier is not freed when using DE_LIST if the CONN_ID parameter was specified on the GET or the GET_ALL functions.

CONN_INTENT={NONE|HOLD}
Specifies the intent of the connection to be used by the GET and GET_ALL functions when a connection is requested.

Intent
Result

NONE
No connection is to be established.

HOLD
Minimal connection to preserve access to the member (or system key/supervisor state only).

This parameter is required by the GET function since a connect intent of NONE is not valid. CONN_INTENT=HOLD must be specified for the GET function.

This parameter is optional and defaults to a connect intent of NONE when used with the GET_ALL function. CONN_INTENT=HOLD for the GET_ALL function requires the caller to be in supervisor state or system key.

CONN_INTENT is meaningful only when one or more of the designated libraries are PDSEs.

DCB=*data_control_block*
specifies the DCB that identifies the libraries to be used for the particular function. *data_control_block* is an open data control block.

For the RENAME, DELETE, and UPDATE functions the DCB must be open for OUTPUT or UPDAT. For all other functions the DCB must be open for INPUT, OUTPUT, or UPDAT.

DE_LIST=(*input_list*,*input_list_entry_count*)
specifies a list of directory entries that identify connections to members that the RELEASE function is to release. The storage must be addressable in the key of the caller.

input_list A–type address. Standard form RX–type address or (2-12). Execute form A–type address. List form.

input_list specifies a list of entries mapped by the DESL structure.

input_list_entry_count Absexp or (2-12). Standard or execute form Absexp. List form

input_list_entry_count contains the number of entries in the list.

For the RELEASE function, DE_LIST is mutually exclusive with CONN_ID.

ENTRY_GAP=({*gap_size*|0})
specifies space to be reserved by the GET and GET_ALL functions within each buffer entry for use by the caller.

gap_size Absexp or (2-12). Standard or Execute form Absexp. List form

gap_size is a numeric value from 0 to 2048.

DESERV places the length specified in the header area of the DESB.

EXT_ATTR={YES | NO}
indicates whether the GET or GET_ALL functions should return the extended attributes in the SMDE. This function is valid only for data member PDSEs.

EXT_ATTR=YES indicates that the returned SMDE will contain the extended attributes, SMDE_EXT_ATTR.

EXT_ATTR=No , the default, indicates that the returned SMDE will not contain the extended attributes.

This is an optional parameter to the GET or the GET_ALL functions.

FUNC={DELETE|GET|GET_ALL|GET_NAMES|RELEASE|RENAME|UPDATE}
specifies the particular function to be performed.

HIDE={YES| NO}
is used for the GET_ALL function to indicate if hidden names are to be visible in the name search. Hidden names are names generated by the program management binder when you specify ALIASES(ALL)

Hidden names are normally used only for program management binding purposes, and are supported only for program objects in PDSE libraries. As a single program object can contain many hidden names and as these names do not represent executable entry points into a module, they are of little interest to end users. Utilities and other programs which list or display member names and aliases typically omit hidden aliases.

YES
DESERV searches for and returns only exposed names (names specified during program management binding).

NO
DESERV searches for and returns all names types.

NO is the default for the HIDE parameter.

MF={L | (E,*parm_list*[,NOCHECK| COMPLETE])| S}
specifies how the macro should generate its code.

L
specifies the list form of the macro. This form generates an inline parameter list, initializes the eye catcher, length, level of parameter, and optionally, sets some static parameters.

E
specifies the execute form of the macro. This form updates a parameter list and transfers control to the service routine.

The third argument, COMPLETE or NOCHECK, is optional. The default is COMPLETE. This argument specifies whether required keyword checking is to be done. If MF=E is coded with the NOCHECK argument, the macro does not check that all required keywords have been specified. If MF=E is coded with the COMPLETE argument (or COMPLETE is allowed to default) the parameter list is cleared to binary zeros (except the header portion, the first 16 (X'10') bytes), and checking is done for all required parameters.

S
specifies the standard form of the macro. This form generates a complete inline expansion of the parameter list, checks for all required and invalid keywords, and invokes the specified function. It should not be used in refreshable or reentrant code sections.

parm_list—RX-type Address or (1-12)

specifies the address of the parameter list. Valid for the MF=E form of the DESERV macro only.

NAME=*name_record*A—type address or (2-12). Standard form.RX—type address or (2–12). Execute formA—type address. List form
specifies the member name on the GET_NAMES function. *name_record* is a varying length byte string of at most 1024 bytes of data. The structure is mapped by the DESN mapping in the IGWDES mapping macro.

name_record specifies either the primary or any of the alias names when used for the GET_NAMES function.

NAME_LIST=(*input_list*,*input_list_entry_count*)
is used with the GET, DELETE, RENAME, UPDATE functions. For GET, it defines the names for which directory entries are to be obtained and points to the output directory entries. For DELETE, it defines the names which are to be deleted. For RENAME, it defines the old names and the new names. For UPDATE, it defines the directory entries which are to be updated.

NAME_LIST is mutually exclusive with the PDSDE parameter.

input_list
A–type address or (2-12). Standard form.
RX-type address or (2–12). Execute form
A-type addres. List form

input_list specifies a list of entries.

The *input_list* structure is mapped by the DESL mapping in the IGWDES mapping macro.

input_list_entry_count
Absexp or (2-12). Standard or execute form.
Absexp. List form.

input_list_entry_count contains the number of entries in the list.

PDSDE=*BLDL_directory_entry* A-type address or (2-12). Standard form Rx-type address or (2-12). Execute form A-type address. List form
specifies a BLDL format directory entry to be used by the GET function to obtain a connection to PDSE member. The member locator token (MLT) for a PDSE member, and concatenation number in the directory entry are used to identify the member. If the concatenation number identifies a PDS, the input BLDL directory entry is converted to SMDE format without searching any directories.

PDSDE is mutually exclusive with the NAME_LIST parameter.

Note: The BLDL directory entry must point to the name portion of a BLDL directory entry, not the FF portion. If the BLDL directory entry points to the FF portion, an RC=0 is returned, but the results are incorrect.

RETCODE=*return_code* A-type address or (2-12). Standard form Rx-type address or (2-12). Execute form A-type address. List form
specifies the name of the variable where the function is to store the return code associated with the result of the function invocation. *return_code* is a four byte value. Independently of whether you code RETCODE, the return code is returned in register 15.

See “DESERV completion codes” on page 258 for valid return code values.

RSNCODE=*reason_code* A-type address or (2-12). Standard form Rx-type address or (2-12). Execute form A-type address. List form
specifies the name of the variable where the function is to store the reason code associated with the result of the function invocation. The high order two bytes of the reason code contain the component id (x'27') and the module identifier of the module which detected the error. The low order two bytes of the reason code contain the actual reason code values. Independently of whether you code RETCODE, the reason code is returned in register 0.

See “Reason codes returned by the DESERV macro” on page 259 for reason code values.

SUBPOOL=*subpool_id* Absexp or (2-12). Standard or execute form Absexp. List form
specifies the subpool identifier to be used by the function when acquiring storage for the buffer. *subpool_id* is a value from 0 to 255 that is optional on the GET, GET_ALL, and GET_NAMES functions.

The actual key and subpool used to acquire storage are:

- If the subpool is specified and is not a user subpool and the caller is NOT authorized (KEY or STATE) the request is rejected as an error.
- If the subpool is specified and is not a user subpool and the caller is authorized the storage is obtained with the subpool specified and the caller's key. This technique assumes that the subpool/key combination is valid. An error occurs if the combination is invalid.
- If the subpool is specified and is a user subpool the storage is obtained with the subpool specified in task key.
- If the subpool is NOT specified the storage is obtained with the subpool 0 in task key.
- If the subpool specified is 0 and the caller is executing in key 0, the storage returned is in subpool 250.

DESERV parameters

Table 34 on page 252 and Table 35 on page 253 show the DESERV macro parameters and indicate for each function if the parameter is required, optional, or invalid. The figure applies to the MF=S (standard) forms of the macro, or to the logically merged MF=L and MF=E parameters.

Table 34. DESERV keyword parameters by function

Keyword / FUNC=	GET	GET_ALL	RELEASE	GET_NAMES
AREA	Optional	Invalid	Invalid	Invalid
AREAPTR	Optional	Required	Invalid	Required
BYPASS_LLA	Optional	Invalid	Invalid	Invalid
CONCAT	Invalid	Optional	Invalid	Optional
CONN_ID	Optional	Optional	Optional	Invalid
CONN_INTENT	Required	Optional	Invalid	Invalid
DCB	Required	Required	Required	Required
DE_LIST	Invalid	Invalid	Optional	Invalid
ENTRY_GAP	Optional	Optional	Invalid	Invalid
FUNC	Required	Required	Required	Required
HIDE	Invalid	Optional	Invalid	Invalid
MF	Optional	Optional	Optional	Optional
NAME	Invalid	Invalid	Invalid	Required
NAME_LIST	Optional	Invalid	Invalid	Invalid
PDSDE	Optional	Invalid	Invalid	Invalid
RETCODE	Optional	Optional	Optional	Optional
RSNCODE	Optional	Optional	Optional	Optional
SUBPOOL	Optional	Optional	Invalid	Optional

Table 35. DESERV keyword parameters by function

Keyword / FUNC=	UPDATE	RENAME	DELETE
AREA	Invalid	Invalid	Invalid
AREAPTR	Invalid	Invalid	Invalid
BYPASS_LLA	Invalid	Invalid	Invalid
CONCAT	Invalid	Invalid	Invalid
CONN_ID	Invalid	Invalid	Invalid
CONN_INTENT	Invalid	Invalid	Invalid
DCB	Required	Required	Required
DE_LIST	Invalid	Invalid	Invalid
ENTRY_GAP	Invalid	Invalid	Invalid
FUNC	Required	Required	Required
HIDE	Invalid	Invalid	Invalid
MF	Optional	Optional	Optional
NAME	Invalid	Invalid	Invalid
NAME_LIST	Required	Required	Required
PDSDE	Invalid	Invalid	Invalid
RETCODE	Optional	Optional	Optional
RSNCODE	Optional	Optional	Optional
SUBPOOL	Invalid	Invalid	Invalid

AREA=(*buffer_area*,*buffer_area_size*) ***buffer_area*—MF=S form, RX–type address or (2-12)** ***buffer_area*—MF=L form, A–type address)** ***buffer_area*—MF=E form, RX–type Address or (2-12)**

Specifies an area provided by the caller into which the GET, GET_G, and GET_ALL_G functions store directory entries.

The area is mapped by the DESB DSECT in the IGWDES mapping macro on return from the function. The storage must be modifiable in the key of the caller.

AREA and AREAPTR are mutually exclusive.

If the area is filled before the processing has ended, the request is terminated at that point. The entries in the buffers are valid and connections may have been established.

For the GET_G and GET_ALL_G functions, to retrieve additional entries, a subsequent GET_G or GET_ALL_G request can be made with name_list specifying the generation name within the last generation directory entry returned in the buffer area.

buffer_area_size—Symbol or (2-12)
absexp or (2-12)—Standard or execute form
absexp—List form

buffer_area_size is the length in bytes of the area specified in the *buffer_area* parameter.

Restriction: There is no way to determine, in advance, the exact buffer size required to contain the directory entries on a single request. A formula for length calculation is provided in Figure 6 on page 254.

FORMULA:
buffer_area_size = L'DESB_FIXED + (*input_list_entry_count* * (SMDE_MAXLEN + *gap_size*))

WHERE:

buffer_area_size
is the storage required to hold *input_list_entry_count* number of entries.

DESB_FIXED
is the fixed (header) portion of the buffer. It is a constant defined by the IGWDES macro.

SMDE_MAXLEN
is a constant defined by macro IGWSMDE that defines the current maximum size of a single SMDE entry. This is a very large value because names can be up to 1024 bytes in length.

gap_size
is the value specified by the ENTRY_GAP parameter on the GET or GET_ALL function.

input_list_entry_count
is the value passed on the NAME_LIST parameter for the number of entries in the list or 1 if PDSDE is specified.

Figure 6. Buffer size calculation for GET function.

AREAPTR=*buffer_area_address* A–type address or (2-12). Standard form RX–type address or (2-12). Execute form A–type address. List form
specifies a word where GET, GET_ALL, and GET_NAMES store the address of the first DESB buffer output.

The buffer-area address points to a chain of buffers mapped by the DESB mapping in the IGWDES mapping macro on return from the function.

The subpool number for the storage obtained is placed in the buffer header. See the description of the SUBPOOL keyword for subpool value determination.

It is your responsibility to release the storage using the STORAGE or FREEMAIN macro.

If you issue a DESERV call while running in 24-bit addressing mode, the storage area returned will be below the 16 MB line. If you issue a DESERV call while running in 31-bit addressing mode, the storage returned can be above or below the 16 MB line.

AREAPTR and AREA are mutually exclusive.

BYPASS_LLA={YES | NO}
indicates whether the GET function should bypass LLA's cached directory entries and go only to the current library directory or use LLA's cached directory entries if they are available.

BYPASS_LLA=YES indicates that the LLA cache is not examined. BYPASS_LLA=NO, the default, indicates that the LLA cache is examined before attempting to obtain information directly from the data set.

This is an optional parameter to the GET function.

Currently, the GET_ALL function does not obtain member list from LLA. Therefore, the directory entries come directly from the data set as though BYPASS_LLA=YES were specified.

Tip: Response time is better if the directory entries are obtained from LLA.

CONCAT={*concat_number*|ALL}
specifies the library concatenation.

***concat_number Absexp or (2-12)* Standard and execute form. *Absexp* List form.**
specifies for the GET_ALL and GET_NAMES function the specific library in a concatenation of libraries. DESERV returns all the member names. *concat_number* is a numeric value in the range of 0 to 255.

This is an optional parameter and the default is the first library in the concatenation (that is, 0).

ALL
specifies (for the GET_ALL function only) to return all the names in each PDS or PDSE directory in the concatenation. DESERV returns a list of directory entries which contains a merged list of SMDEs from each data set in the concatenation where duplicate member names have been eliminated.

CONN_ID=*connection_addr* A—type address or (2-12). Standard form RX—type address or (2-12). Execute form A—type address. List form
specifies the location of the 4-byte value used by the GET, GET_ALL, and RELEASE functions. The four bytes are a token that relates connections to a particular invocation of a function. It may indicate a number of connections or no connections at all.

For the RELEASE function, CONN_ID is an input parameter and is mutually exclusive with DE_LIST.

For the GET and GET_ALL functions, CONN_ID is an output parameter.

CONN_ID is meaningful only when one or more of the designated libraries are PDSEs.

Note: A maximum of 65536 connection identifiers per DCB can exist simultaneously. You can free the identifier by using the RELEASE function and specifying the connection identifier to be freed. The identifier is not freed when using DE_LIST if the CONN_ID parameter was specified on the GET or the GET_ALL functions.

CONN_INTENT={NONE|HOLD}
Specifies the intent of the connection to be used by the GET and GET_ALL functions when a connection is requested.

Intent
Result

NONE
No connection is to be established.

HOLD
Minimal connection to preserve access to the member (or system key/supervisor state only).

This parameter is required by the GET function since a connect intent of NONE is not valid. CONN_INTENT=HOLD must be specified for the GET function.

This parameter is optional and defaults to a connect intent of NONE when used with the GET_ALL function. CONN_INTENT=HOLD for the GET_ALL function requires the caller to be in supervisor state or system key .

CONN_INTENT is meaningful only when one or more of the designated libraries are PDSEs.

DCB=*data_control_block*
specifies the DCB that identifies the libraries to be used for the particular function. *data_control_block* is an open data control block.

For the RENAME, DELETE, and UPDATE functions the DCB must be open for OUTPUT or UPDAT. For all other functions the DCB must be open for INPUT, OUTPUT, or UPDAT.

DE_LIST=(*input_list*,*input_list_entry_count*)
specifies a list of directory entries that identify connections to members that the RELEASE function is to release. The storage must be addressable in the key of the caller.

input_list A–type address. Standard form RX–type address or (2-12). Execute form A–type address. List form.

input_list specifies a list of entries mapped by the DESL structure.

input_list_entry_count Absexp or (2-12). Standard or execute form Absexp. List form

input_list_entry_count contains the number of entries in the list.

For the RELEASE function, DE_LIST is mutually exclusive with CONN_ID.

ENTRY_GAP=({*gap_size*|0**})**
specifies space to be reserved by the GET and GET_ALL functions within each buffer entry for use by the caller.

gap_size Absexp or (2-12). Standard or Execute form Absexp. List form

gap_size is a numeric value from 0 to 2048.

DESERV places the length specified in the header area of the DESB.

EXT_ATTR={YES | NO}
indicates whether the GET or GET_ALL functions should return the extended attributes in the SMDE. This function is valid only for data member PDSEs.

EXT_ATTR=YES indicates that the returned SMDE will contain the extended attributes, SMDE_EXT_ATTR.

EXT_ATTR=No , the default, indicates that the returned SMDE will not contain the extended attributes.

This is an optional parameter to the GET or the GET_ALL functions.

FUNC={DELETE|GET|GET_ALL|GET_NAMES|RELEASE|RENAME|UPDATE}
specifies the particular function to be performed.

HIDE={YES| NO}
is used for the GET_ALL function to indicate if hidden names are to be visible in the name search. Hidden names are names generated by the program management binder when you specify ALIASES(ALL)

Hidden names are normally used only for program management binding purposes, and are supported only for program objects in PDSE libraries. As a single program object can contain many hidden names and as these names do not represent executable entry points into a module, they are of little interest to end users. Utilities and other programs which list or display member names and aliases typically omit hidden aliases.

YES
DESERV searches for and returns only exposed names (names specified during program management binding).

NO
DESERV searches for and returns all names types.

NO is the default for the HIDE parameter.

MF={L | E,*parm_list*[,NOCHECK| COMPLETE]| S}
specifies how the macro should generate its code.

L
specifies the list form of the macro. This form generates an inline parameter list, initializes the eye catcher, length, level of parameter, and optionally, sets some static parameters.

E
specifies the execute form of the macro. This form updates a parameter list and transfers control to the service routine.

The third argument, COMPLETE or NOCHECK, is optional. The default is COMPLETE. This argument specifies whether required keyword checking is to be done. If MF=E is coded with the NOCHECK argument, the macro does not check that all required keywords have been specified. If MF=E is coded with the COMPLETE argument (or COMPLETE is allowed to default) the parameter list is cleared to binary zeros (except the header portion, the first 16 (X'10') bytes), and checking is done for all required parameters.

S
specifies the standard form of the macro. This form generates a complete inline expansion of the parameter list, checks for all required and invalid keywords, and invokes the specified function. It should not be used in refreshable or reentrant code sections.

parm_list—RX-type Address or (1-12)

specifies the address of the parameter list. Valid for the MF=E form of the DESERV macro only.

NAME=*name_record*A—type address or (2-12). Standard form.RX—type address or (2–12). Execute formA—type address. List form
specifies the member name on the GET_NAMES function. *name_record* is a varying length byte string of at most 1024 bytes of data. The structure is mapped by the DESN mapping in the IGWDES mapping macro.

name_record specifies either the primary or any of the alias names when used for the GET_NAMES function.

NAME_LIST=(*input_list*,*input_list_entry_count*)
is used with the GET, GET_G, GET_ALL_G, DELETE, RENAME, UPDATE functions.

For GET, it defines the names for which directory entries are to be obtained and points to the output directory entries.

For GET_G, it defines a primary name for which generation directory entries are to be obtained. input_list_entry_count must indicate one entry. If the output area is filled before all generation directory entries could be returned, subsequent calls can be made to return additional generation directory entries for the initial primary name. On these subsequent calls, the NAME_LIST should contain the generation name (12-byte SMDE_GENE_NAME) to be used as a continuation point. It can be copied from the last generation directory entry found in the output area returned by the previous call. Input_list_entry_count must indicate one entry.

For GET_ALL_G, NAME_LIST is not required to return generation directory entries for a PDSE. However, if the output area is filled before all generation directory entries could be returned, subsequent calls can be made using NAME_LIST to return additional generation directory entries for the PDSE. On these subsequent calls, the AME_LIST should contain the generation name (12-byte SMDE_GENE_NAME) to be used as a continuation point. It can be copied from the last generation directory entry found in the output area returned by the previous call. Input_list_entry_count must indicate one entry.

For DELETE, it defines the names which are to be deleted. For RENAME, it defines the old names and the new names.

For UPDATE, it defines the directory entries which are to be updated.

NAME_LIST is mutually exclusive with the PDSDE parameter.

input_list
A–type address or (2-12). Standard form.
RX-type address or (2–12). Execute form
A-type addres. List form

input_list specifies a list of entries.

The *input_list* structure is mapped by the DESL mapping in the IGWDES mapping macro.

input_list_entry_count
Absexp or (2-12). Standard or execute form.
Absexp. List form.

input_list_entry_count contains the number of entries in the list.

PDSDE=*BLDL_directory_entry* A-type address or (2-12). Standard form Rx-type address or (2-12). Execute form A-type address. List form
specifies a BLDL format directory entry to be used by the GET function to obtain a connection to PDSE member. The member locator token (MLT) for a PDSE member, and concatenation number in the directory entry are used to identify the member. If the concatenation number identifies a PDS, the input BLDL directory entry is converted to SMDE format without searching any directories.

PDSDE is mutually exclusive with the NAME_LIST parameter.

Note: The BLDL directory entry must point to the name portion of a BLDL directory entry, not the FF portion. If the BLDL directory entry points to the FF portion, an RC=0 is returned, but the results are incorrect.

RETCODE=*return_code* A-type address or (2-12). Standard form Rx-type address or (2-12). Execute form A-type address. List form
specifies the name of the variable where the function is to store the return code associated with the result of the function invocation. *return_code* is a four byte value. Independently of whether you code RETCODE, the return code is returned in register 15.

See "DESERV completion codes" on page 258 for valid return code values.

RSNCODE=*reason_code* A-type address or (2-12). Standard form Rx-type address or (2-12). Execute form A-type address. List form
specifies the name of the variable where the function is to store the reason code associated with the result of the function invocation. The high order two bytes of the reason code contain the component id (x'27') and the module identifier of the module which detected the error. The low order two bytes of the reason code contain the actual reason code values. Independently of whether you code RETCODE, the reason code is returned in register 0.

See "Reason codes returned by the DESERV macro" on page 259 for reason code values.

SUBPOOL=*subpool_id* Absexp or (2-12). Standard or execute form Absexp. List form
specifies the subpool identifier to be used by the function when acquiring storage for the buffer. *subpool_id* is a value from 0 to 255 that is optional on the GET, GET_ALL, and GET_NAMES functions.

The actual key and subpool used to acquire storage are:

- If the subpool is specified and is not a user subpool and the caller is NOT authorized (KEY or STATE) the request is rejected as an error.
- If the subpool is specified and is not a user subpool and the caller is authorized the storage is obtained with the subpool specified and the caller's key. This technique assumes that the subpool/key combination is valid. An error occurs if the combination is invalid.
- If the subpool is specified and is a user subpool the storage is obtained with the subpool specified in task key.
- If the subpool is NOT specified the storage is obtained with the subpool 0 in task key.
- If the subpool specified is 0 and the caller is executing in key 0, the storage returned is in subpool 250.

DESERV completion codes

The DESERV macro *return codes* with their descriptions are shown below, followed by the *reason codes*. The *reason codes* are grouped by DESERV macro function.

When the system returns control to the problem program, the return code is in the area identified by the RETCODE= parameter or register 15 and the reason code is in the area identified by the RSNCODE= parameter or register 0. The significant part of the reason code is in the low-order 2 bytes.

The symbols included in the descriptions below for the return and reason codes (for example, DESRC_SUCC) are contained in the macro IGWDES.

A system error return code (DESRC_SEVE (36(X'24'))) should be considered to be a terminating error. The reason codes associated with DESRC_SEVE are for Diagnosis, Modification, and Tuning Information (DMTI) and are contained in *z/OS DFSMSdfp Diagnosis*.

Return codes returned by the DESERV macro

Table 36. DESERV Return Codes

Return Code	Name	Meaning
00(X'00')	DESRC_SUCC	Successful processing.
04(X'04')	DESRC_INFO	Not completely successful.
08(X'08')	DESRC_WARN.	Results questionable.
12(X'0C')	DESRC_PARM	Missing or invalid parameters.
16(X'10')	DESRC_CALR	Caller has a problem.
20(X'14')	DESRC_ENVR	Resources unavailable.
24(X'18')	DESRC_IOER	I/O error.
28(X'1C')	DESRC_MEDE	Media error.
32(X'20')	DESRC_DSLE	Data set logical error.
36(X'24')	DESRC_SEVE	System error. See *z/OS DFSMSdfp Diagnosis* for DESERV system codes.

Reason codes returned by the DESERV macro

DESERV reason codes returned from the macro invocation are four byte values. The values listed here are the low order two byte values. The high order two bytes are used for diagnostic purposes and should not be tested by your program.

The reason codes below are separated by DESERV function. The return codes shown in each figure below are described above.

DESERV functions common reason codes

Table 37. DESERV Functions Common Reason Codes

Return Code	Reason Code	Name	Meaning
00(X'00')	0000(X'00')	DESRS_SUCC	Successful processing.
12(X'0C')	1041(X'411')	DESRS_INVALID_PARM_LIST_HEADER	The id, length, or level of the parameter list is invalid.
12(X'0C')	1054(X'41E')	DESRS_INVALID_DEB_PTR	Address of the DEB is 0 or DEB is input but the DCB pointed to by the DEB does not point back to the DEB.
12(X'0C')	1057(X'421')	DESRS_DCB_NOT_OPEN	The passed DCB is not open.
12(X'0C')	1058(X'422')	DESRS_INVALID_DCB_PTR	The address of the DCB is zero.
12(X'0C')	1059(X'423')	DESRS_DEB_REQUIRES_AUTH	To pass the DEB the caller must be supervisor state or a system key.
12(X'0C')	1060(X'424')	DESRS_UNSUPPORTED_FUNC	The FUNC value is incorrect.
16(X'10')	1053(X'41D')	DESRS_DEBCHK_FAILED	The DEBCHK macro failed. The DCB or DEB is invalid.

DESERV GET function reason codes

Table 38. DESERV GET Function Reason Codes

Return Code	Reason Code	Meaning	Name
00(X'00')	0000(X'00')	DESRS_SUCC	Successful processing.
04(X'04')	1001(X'3E9')	DESRS_MODULE_BUFFERED_LLA	The module is buffered by LLA, no connection is established.
04(X'04')	1002(X'3EA')	DESRS_NOTFOUND	Some members not found.
04(X'04')	1020(X'3FC')	DESRS_CANT_GET_FILELOCK	File lock unavailable, possible sharing problem.
12(X'0C')	1003(X'3EB')	DESRS_C370LIB_SMDE_ME	The SMDE parameter is mutually exclusive with C370LIB(YES).
12(X'0C')	1004(X'3EC')	DESRS_SMDE_PTR_INVALID	For GETTYPE=SMDE, the input pointer is zero.
12(X'0C')	1005(X'3ED')	DESRS_AREA_AREAPTR_ME	AREA and AREAPTR are mutually exclusive.
12(X'0C')	1010(X'3F2')	DESRS_C370LIB_PDSDE_ME	C370LIB(YES) and PDSDE are mutually exclusive.
12(X'0C')	1051(X'41B')	DESRS_PDSDE_PTR_INVALID	Address of the PDSDE is 0.
12(X'0C')	1070(X'42E')	DESRS_INVALID_ENTRY_GAP	The gap specified is too large. This gap must be no larger than DESP_ENTRY_GAP_MAX.
12(X'0C')	1071(X'42F')	DESRS_AREA_LENGTH_TOO_SMALL	The length of the area provided is insufficient. For the GET function this area length must be larger than the fixed portion of the DESB.
12(X'0C')	1073(X'431')	DESRS_INVALID_AREA_PTR	The address of a DESB provided is 0.
12(X'0C')	1074(X'432')	DESRS_INVALID_GETTYPE	The GET function accepts only a NAME_LIST or a PDSDE. Neither is provided.
12(X'0C')	1076(X'434')	DESRS_NAME_LIST_COUNT_INVALID	The count of entries in the NAME_LIST is 0.
12(X'0C')	1077(X'435')	DESRS_NAME_LIST_@_INVALID	The address of the NAME_LIST structure is 0.
12(X'0C')	1078(X'436')	DESRS_INVALID_CONN_INTENT	The connect intent specified is not valid with this function.
16(X'10')	1006(X'3EE')	DESRS_DCB_NOT_OPEN_PO	The DCB is not opened with DSORG=PO. This applies only to the GET function when C370LIB(YES).
16(X'10')	1009(X'3F1')	DESRS_BAD_BLKSIZE	DCBBLKSI is too small.
16(X'10')	1046(X'416')	DESRS_INSUF_BUFFER_SIZE	Area provided is too small.

Table 38. DESERV GET Function Reason Codes (continued)

Return Code	Reason Code	Meaning	Name
16(X'10')	1061(X'425')	DESRS_INVALID_NAME_LENGTH	The length of an alias name is either 0 or greater than 1024. The length of a primary name is 0 or greater than 8.
20(X'14')	1035(X'40B')	DESRS_FREEMAIN_ERROR	FREEMAIN failed.
24(X'18')	1034(X'40A')	DESRS_CONVERT_ERROR	Error converting TTR to CCHHR.
24(X'18')	1086(X'43E')	DESRS_ECB_POSTED_ERROR	An I/O error was received, the post code in the ECB was unexpected.
32(X'20')	1007(X'3EF')	DESRS_BAD_C370LIB_DIR	The C370LIB directory indicates that a symbol is associated with a member name but that name does not exist in the PDS directory.
32(X'20')	1008(X'3F0')	DESRS_BAD_TXT_CARD	Inconsistencies found in the text records, while processing a C370LIB directo

DESERV GET_ALL function reason codes

Table 39. DESERV GET_ALL Function Reason Codes

Return Code	Reason Code	Meaning	Name
00(X'00')	0000(X'00')	DESRS_SUCC	Successful processing.
08(X'08')	1012(X'3F4')	DESRS_DIRECTORY_EMPTY	No members in directory.
12(X'0C')	1045(X'415')	DESRS_PDS_NOT_SUPPORTED	This function requires a PDSE data set.
12(X'0C')	1052(X'41C')	DESRS_INVALID_CONCAT	The concatenation number specified is greater than the concatenation number of the last data set in the concatenation.
12(X'0C')	1070(X'42E')	DESRS_INVALID_ENTRY_GAP	The gap specified is too large. The gap must be larger than DESP_ENTRY_GAP_MAX.
12(X'0C')	1072(X'430')	DESRS_INVALID_AREAPTR_PTR	The address of the AREAPTR is 0.
12(X'0C')	1078(X'436')	DESRS_INVALID_CONN_INTENT	The connect intent specified is not valid with this function.
16(X'10')	1011(X'3f3')	DESRS_CONN_AUTH	The CONN_INTENT(HOLD) requires the caller of function GET_ALL to be in supervisor state or system key.
20(X'14')	1035(X'40B')	DESRS_FREEMAIN_ERROR	FREEMAIN failure.

DESERV GET_ALL_G function reason codes

Table 40. DESERV GET_ALL_G Function Reason Codes

Return Code	Reason Code	Meaning	Name
00(X'00')	0000(X'00')	DESRS_SUCC	Successful processing.
12(X'0C')	1131(X'46B')	DESRS_NAME_LIST_LENGTH_NOT12	The Name_List parm supplied was not the required 12 bytes in length.
16(X'10')	1046(X'416')	DESRS_INSUF_BUFFER_SIZE	Area provided is too small.
16(X'10')	1130(X'46A')	DESRS_CONCAT_NOT_ONE	The PDSE is in a concatenation. GET_ALL_G does not support a PDSE concatenation.

DESERV GET_G function reason codes

Table 41. DESERV GET_G Function Reason Codes

Return Code	Reason Code	Meaning	Name
00(X'00')	0000(X'00')	DESRS_SUCC	Successful processing.
16(X'10')	1046(X'416')	DESRS_INSUF_BUFFER_SIZE	Area provided is too small.
16(X'10')	1130(X'46A')	DESRS_CONCAT_NOT_ONE	The PDSE is in a concatenation. GET_G does not support a PDSE concatenation.

DESERV GET_NAMES function reason codes

Table 42. DESERV GET_NAMES Function Reason Codes

Return Code	Reason Code	Name	Meaning
00(X'00')	0000(X'00')	DESRS_SUCC	Successful processing.
08(X'08')	1002(X'3EA')	DESRS_NOTFOUND	Some members not found.
12(X'0C')	1045(X'415')	DESRS_PDS_NOT_SUPPORTED	This function requires a PDSE data set.
12(X'0C')	1052(X'41C')	DESRS_INVALID_CONCAT	The concatenation number specified is greater than the concatenation number of the last data set.
12(X'0C')	1061(X'425')	DESRS_INVALID_NAME_LENGTH	The length of an alias name is either 0 or greater than 1024, or the length of a primary name is 0 or greater than 8.
12(X'0C')	1062(X'426')	DESRS_INVALID_NAME_PTR	The address of the NAME parameter is 0.
12(X'0C')	1072(X'430')	DESRS_INVALID_AREAPTR_PTR	The address of the AREAPTR is 0.
20(X'14')	1035(X'40B')	DESRS_FREEMAIN_ERROR	FREEMAIN failure.

DESERV RELEASE function reason codes

Table 43. DESERV RELEASE Function Reason Codes

Return Code	Reason Code	Meaning	Name
00(X'00')	0000(X'00')	DESRS_SUCC	Successful processing.
08(X'08')	1019(X'3FB')	DESRS_CONNECTION_NOT_FOUND	The connection specified in the SMDE could not be found. Probable user error.
12(X'0C')	1066(X'42A')	DESRS_INVALID_RELEASE_TYPE	The RELEASE function must be specified with the CONN_ID parameter or the DE_LIST parameter.
12(X'0C')	1067(X'42B')	DESRS_INVALID_CONN_ID_PTR	The address of the CONN_ID parameter is 0.
12(X'0C')	1068(X'42C')	DESRS_INVALID_DE_LIST_CNT	The number of entries in the DE_LIST is 0.
12(X'0C')	1069(X'42D')	DESRS_INVALID_DE_LIST_PTR	The address of the DE_LIST parameter is 0.
16(X'10')	1018(X'3FA')	DESRS_DESL_SMDE_PTR	The SMDE for the release function had a null pointer or the eye catcher is invalid.

DESERV UPDATE function reason codes

Table 44. DESERV UPDATE Function Reason Codes

Return Code	Reason Code	Meaning	Name
00(X'00')	0000(X'00')	DESRS_SUCC	Successful processing.
08(X'08')	1002(X'3EA')	DESRS_NOTFOUND	Some members not found
08(X'08')	1014(X'3F6')	DESRS_MULTIPLE_ERRORS	More than one error has occurred. Check the codes in DESL.
12(X'0C')	1045(X'415')	DESRS_PDS_NOT_SUPPORTED	This function requires a PDSE data set.
12(X'0C')	1062(X'426')	DESRS_INVALID_NAME_PTR	The address of the parameter is 0.
12(X'0C')	1076(X'434')	DESRS_NAME_LIST_COUNT_INVALID	The count of entries in the NAME_LIST is 0.
12(X'0C')	1077(X'435')	DESRS_NAME_LIST_@_INVALID	The address of NAME_LIST structure is 0
16(X'10')	1061(X'425')	DESRS_INVALID_NAME_LENGTH	The length of a name is either 0 or greater than 8.

DESERV DELETE function reason codes

Table 45. DESERV DELETE Function Reason Codes

Return Code	Reason Code	Meaning	Name
00(X'00')	0000(X'00')	DESRS_SUCC	Successful processing.

Table 45. DESERV DELETE Function Reason Codes (continued)

Return Code	Reason Code	Meaning	Name
08(X'08')	1002(X'3EA')	DESRS_NOTFOUND	One member not found.
08(X'08')	1014(X'3F6')	DESRS_MULTIPLE_ERRORS	More than one error occurred. Check the codes in DESL.
12(X'0C')	1045(X'415')	DESRS_PDS_NOT_SUPPORTED	This function requires a PDSE data set
12(X'0C')	1062(X'426')	DESRS_INVALID_NAME_PTR	The address of the parameter is 0
12(X'0C')	1076(X'434')	DESRS_NAME_LIST_COUNT_INVALID	The count of entries in the NAME_LIST is 0.
12(X'0C')	1077(X'435')	DESRS_NAME_LIST_@_INVALID	The address of a NAME_LIST structure is 0
16(X'10')	1061(X'425')	DESRS_INVALID_NAME_LENGTH	The length of a name is either 0 or greater than 8.

DESERV RENAME function reason codes

Table 46. DESERV RENAME Function Reason Codes

Return Code	Reason Code	Meaning	Name
00(X'00')	0000(X'00')	DESRS_SUCC	Successful processing.
08(X'08')	1002(X'3EA')	DESRS_NOTFOUND	Some members not found
08(X'08')	1014(X'3F6')	DESRS_MULTIPLE_ERRORS	More than one error occurred. Check the codes in DESL.
08(X'08')	1040(X'410')	DESRS_INVALID_NAME_PREFIX	The first 8 bytes of the name were all X'FF'.
08(X'08')	1108(X'454')	DESRS_BOTH_NAMES_SAME	A FUNC=RENAME request specified a new name and an old name which were the sam
08(X'08')	1110(X'456')	DESRS_NEW_NAME_EXISTS	A FUNC=RENAME request specified a new name which already exists in the pdse.
12(X'0C')	1045(X'415')	DESRS_PDS_NOT_SUPPORTED	This function request is not for a PDSE data set
12(X'0C')	1062(X'426')	DESRS_INVALID_NAME_PTR	The address of the name parameter is 0.
12(X'0C')	1076(X'434')	DESRS_NAME_LIST_COUNT_INVALID	The count of entries in the NAME_LIST is 0.
12(X'0C')	1077(X'435')	DESRS_NAME_LIST_@_INVALID	The address of NAME_LIST structure is 0
16(X'10')	1061(X'425')	DESRS_INVALID_NAME_LENGTH	The length of a name is either 0 or greater than 8.
20(X'14')	1083(X'43B')	DESRS_CLOCK_ERROR	An STCK instruction failed.

ESETL—End sequential retrieval (QISAM)

The ESETL macro ends the sequential retrieval of data from an indexed sequential data set and causes the buffers associated with the specified data control block to be released. An ESETL macro must separate SETL macros issued for the same data control block.

Recommendation: Do not use the ESETL macro because it is a QISAM macro. Instead, use VSAM.

The format of the ESETL macro is:

[*label*]	ESETL	*dcb address*

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block that is opened for the indexed sequential data set being processed.

FEOV—Force end-of-volume (BSAM and QSAM)

The FEOV macro causes the system to assume an end-of-volume condition, and switches volumes automatically. You can specify volume positioning for magnetic tape with the REWIND or LEAVE option. If no option is coded, the positioning specified in the OPEN macro is used. Output labels are created as required and new input labels are verified. The standard exit routines are given control as specified in the data control block exit list. For BSAM, you must test all input and output operations for completion before issuing the FEOV macro. The end-of-data (EODAD) routine is given control if an input FEOV macro is issued for the last volume of an input data set and another data set is not concatenated.

If the current data set is part of a concatenation and you are on the last or only volume, the system switches to the next data set. If you are at the end of the last data set, the end-of-data routine is given control. If the EODAD routine is needed but you did not specify one, the FEOV issues ABEND 337-04.

FEOV is ignored if issued for a SYSIN or SYSOUT data set or if the data set is closed.

FEOV treats an UNIX file or a striped data set as a single volume data set which cannot be extended to additional volumes.

The FEOV macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses. If it causes entry to the end-of-data (EODAD) routine, the EODAD routine is entered in the addressing mode in which you issue FEOV.

During FEOV processing, if an error occurs that is ignored by the user's DCB ABEND exit, the DCB will be closed on return from FEOV. It is recommended that the DCB be checked to verify it is still open upon return from FEOV before issuing any other macro using that DCB other than CLOSE or FREEPOOL.

Recommendation: Issue an FEOV in 31-bit addressing mode when processing a DCB open for output that specifies QSAM locate mode and the buffers are above the 16MB line (DCBE RMODE31=BUFF is specified), .

The format of the FEOV macro is:

[*label*]	FEOV	*dcb address* `[,REWIND|,LEAVE]`

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for an opened sequential data set.

REWIND
requests that the system position the tape that you are leaving at the load point regardless of the direction of processing.

LEAVE
requests that the system position the tape at the logical end of the data set on the volume that you are leaving. This option positions the tape at a point after the tape mark that follows the trailer labels. Multiple tape units must be available to achieve this positioning. If only one tape unit is available, its volume is rewound and unloaded.

Restrictions are as follows:

- If an FEOV macro is issued for a multivolume data set with spanned records that is being read using QSAM, errors might occur when the next GET macro is issued following an FEOV macro if the first segment on the new volume is not the first segment of a record. The errors include duplicate records, program checks in your user program, and invalid input from the variable spanned data set.
- Do not use the FEOV macro in the error analysis routine (SYNAD).

FIND—Establish the beginning of a data set member (BPAM)

The FIND macro causes the system to use the address of the first block of a specified partitioned data set member as the starting point for the next READ macro for the same data set. All previous input and output operations that specified the same data control block must have been tested for completion before the FIND macro is issued.

When used with a PDSE, the FIND macro establishes a connection to a PDSE member. If FIND by relative address (C option) was specified, the connection remains until the PDSE is closed. If FIND by name (D option) was specified, the connection remains until you position to another member.

If the PDSE is open for output, close it and reopen it for input or update processing before issuing the FIND macro. See *z/OS DFSMS Using Data Sets* for more information on using the FIND macro and PDSE connections.

You can issue the FIND macro in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses. If the DCB points to a DCBE that resides above the 16MB line, you must issue the FIND macro in 31-bit addressing mode.

The format of the FIND macro is:

[*label*]	FIND	*dcb address* ,{*name address*,D|*ttrc address*,C|*generation plist*,G}

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for the opened partitioned data set being processed.

***name address*—RX-Type Address, (2-12), or (0)**
specifies the address of an 8-byte field that contains the data set member name. The name must start in the first byte and be padded on the right (if necessary) to complete the 8 bytes. The name address may point above or below the 16MB line.

D
specifies that only a member name has been supplied, and the access method must search the directory of the data set indicated in the data control block to find the location of the member.

***ttr address*—RX-Type Address, (2-12), or (0)**
specifies the address of a 4-byte area that contains the 3-byte relative address (TTR) and a 1-byte concatenation number (C). The TTRC address can point to the TTRC field in a BLDL list entry completed by using a BLDL macro for the data set being processed.

C
specifies that a TTRC address has been supplied, and no directory search is required. The TTRC address supplied is used directly by the access method for the next input operation.

***generation plist*—RX-Type Address, (2-12), or (0)**

Offset	Length	Contents
X'00'	2	Length of plist=20
X'02'	1	Must be zero
X'03'	1	Must be zero
X'04'	4	Length of the member name
X'08'	8	Member name
X'10'	4	Generation number (returned by DESERV) or a negative relative generation number

G
specifies that a member name and generation are being passed and that the records from the generation should be returned.

Rule: Do not use the FIND macro after WRITE and STOW processing without first closing the data set and reopening it for INPUT processing.

FIND completion codes

For *ttrc address*, C, when the system returns control to the problem program, the contents of register 15 are set to 0. If the TTRC address is in error, execution of the next CHECK macro causes control to be passed to the error analysis (SYNAD) routine.

For *name address*, D, when the system returns control to the problem program, the 3 high-order bytes of registers 0 and 15 are set to 0, the low-order byte of register 15 contains one of the following return codes and the low-order byte of register 0 contains one of the following reason codes:

Table 47. FIND Completion Codes

Return Code (15)	Reason Code (0)	Meaning
00 (X'00')	00 (X'00')	Successful execution.
04 (X'04')	00 (X'00')	Name not found.
04 (X'04')	04 (X'04')	The caller has only RACF execute authority to the PDSE.
04 (X'04')	04 (X'04')	An attempt to connect to a UNIX file failed because the some data set in the concatenation is protected with RACF execute-only authority.
04 (X'04')	08 (X'08')	The PDSE member's share options do not allow you to access it.
04 (X'04')	12 (X'0C')	The PDSE is open for output and the FIND macro was issued to point to a member other than the one currently processing.
08 (X'08')	00 (X'00')	Permanent I/O error during directory search.
08 (X'08')	04 (X'04')	Insufficient virtual storage available.
08 (X'08')	08 (X'08')	Invalid DEB, or DEB is not owned by a TCB in the current family of TCBs.
08 (X'08')	12 (X'0C')	An I/O error occurred while flushing system buffers containing member data (PDSE only).
08 (X'08')	16 (X'10')	No DCB address was input.
08 (X'08')	20 (X'14')	An error was returned by IGGSOOPN when attempting to connect to a UNIX file. See message IEC104I for more details.

Table 47. FIND Completion Codes (continued)

Return Code (15)	Reason Code (0)	Meaning
08 (X'08')	24 (X'18')	An attempt to connect to a UNIX file failed because the user did not have RACF authority to access to the file.

FREEBUF—Return a buffer to a pool (BDAM, BISAM, BPAM, and BSAM)

The FREEBUF macro causes the system to return a buffer to the buffer pool assigned to the specified data control block. The buffer must have been acquired using a GETBUF macro.

The FREEBUF macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses. FREEBUF does not support buffers above the line.

The format of the FREEBUF macro is:

[*label*]	FREEBUF	*dcb address* ,*register*

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for an opened data set to which the buffer pool has been assigned. When issued in 31-bit addressing mode, the input DCB address and buffer address must be clean 31-bit addresses.

***register*—(2-12)**
specifies one of registers 2 through 12 that contains the address of the buffer being returned to the buffer pool.

FREEDBUF—Return a dynamically obtained buffer (BDAM and BISAM)

The FREEDBUF macro causes the system to return a buffer to the buffer pool assigned to the specified data control block. The buffer must have been acquired through dynamic buffering; that is, by coding 'S' for the *area address* in the associated READ macro. FREEDBUF does not support buffers above the line.

A buffer acquired dynamically can also be released by a WRITE macro. See the description of the WRITE macro for BDAM or BISAM.

The FREEDBUF macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses. Both FREEDBUF parameters must reside below the 16MB line, so FREEDBUF will ignore the high-order bytes of their addresses.

The format of the FREEDBUF macro is:

[*label*]	FREEDBUF	*decb address* ,{K|D} ,*dcb address*

***decb address*—RX-Type Address, (2-12), or (0)**
specifies the address of the data event control block (DECB) used or created by the READ macro that acquired the buffer dynamically. When issued in 31-bit addressing mode, the buffers must reside below the 16MB line.

K
specifies that BISAM is being used.

D
specifies that BDAM is being used.

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for the opened data set being processed.

FREEPOOL—Release a buffer pool (BDAM, BISAM, BPAM, BSAM, QISAM, and QSAM)

The FREEPOOL macro releases an area of storage, previously acquired for a buffer pool for a specified data control block. The area must have been acquired either automatically (except when dynamic buffer control is used) or by executing a GETPOOL macro. For queued access methods, you must issue a CLOSE macro for all the data control blocks using the buffer pool *before* issuing the FREEPOOL macro. For basic access methods, you can issue the FREEPOOL macro when the buffers are no longer required. A buffer pool need be released only once, regardless of the number of data control blocks sharing the buffer pool.

The FREEPOOL macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

When you request that OPEN obtain QSAM buffers above the 16MB line by coding RMODE31=BUFF on the DCBE macro, CLOSE will free the buffer pool.

If you issue a FREEPOOL macro for a DCB that does not have a buffer pool, the FREEPOOL has no effect.

FREEPOOL does not support buffers above the line.

The format of the FREEPOOL macro is:

[*label*]	FREEPOOL	*dcb address*

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of a data control block to which the buffer pool is assigned. When issued in 31-bit addressing mode, the input DCB address must be a clean 31-bit address.

GET—Obtain next logical record (QISAM)

The GET macro retrieves (reads) the next record. Control is not returned to the problem program until the record is available.

The format of the GET macro is:

[*label*]	GET	*dcb address* [,*area address*]

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for the opened input data set being retrieved.

***area address*—RX-Type Address, (2-12), or (0)**
specifies the storage address into which the system is to move the record (move mode only). Either the move or locate mode can be used with QISAM, but they must not be mixed in the specified data control block. The following describes operations for move and locate modes:

Locate Mode: If locate mode is specified in the data control block, the *area address* must be omitted. The system returns the address of the buffer segment containing the record in register 1.

Move Mode: If move mode is specified in the data control block, the *area address* must specify the address in the problem program into which the system will move the record. If the *area address* is

omitted, the system assumes that register 0 contains the area address. When control is returned to the problem program, register 0 contains the area address, and register 1 contains the address of the data control block.

Note:

1. The end-of-data-set (EODAD) routine is given control if the end of the data set is reached. The data set can be closed if processing is completed, or an ESETL macro must be issued before a SETL macro to continue further input processing.
2. The error analysis (SYNAD) routine is given control if the input operation could not be completed successfully. The contents of the general registers when control is given to the SYNAD user exit routine are described in *z/OS DFSMS Using Data Sets*.
3. When the key of an unblocked record is retrieved with the data, the address of the key is returned as follows (see the SETL macro):

   Locate Mode: The address of the key is returned in register 0.

   Move Mode: The key appears before the record in your buffer area.
4. If a GET macro is issued for a data set and the previous request issued for the same data set was an OPEN, ESETL, or unsuccessful SETL (no record found), a SETL B (key and data) is invoked automatically, and the first record in the data set is returned.

GET—Obtain next logical record (QSAM)

The GET macro retrieves (reads) the next record. Various modes are available and are specified in the DCB macro.

In the locate mode, the GET macro instruction locates the next sequential record or record segment to be processed. The system returns the address of the record or segment in register 1. If you are reading undefined-length records (RECFM=U) and you are not using the large block interface (LBI), the system places the length of the record or segment in the logical record length (DCBLRECL) field of the data control block. The DCBLRECL field is not changed when GET is used in XLRI processing. You can process the record in the input buffer or move the record to a work area.

If you are using the large block interface (OPEN set DCBESLBI on) for undefined-length records, the actual length of the record block that was read is in a 4-byte length-read field. Find the length-read field as follows:

1. After return from the GET macro, and before issuing any other macros against this DCB, obtain the address in DCBIOBA, which is the word at offset X'44' in the DCB.
2. Subtract 4 from the address to obtain the address of the 4-byte length-read field.

In move mode, the GET macro moves the next sequential record to your work area. This work area must be large enough to contain the largest logical record of the data set and its record-descriptor word (variable-length records). The system returns the address of the work area in register 1. The record length is placed in the DCBLRECL field. You can use move mode only with simple buffering.

In data mode, which is available only for variable-length spanned records, the GET macro moves only the data portion of the next sequential record to your work area. You cannot use the TYPE=P parameter with data mode.

The GET macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses. This includes allowing the caller to issue QSAM macros in 31-bit addressing mode regardless of whether the buffers are above or below the 16MB line. Most types of data sets support 31-bit mode. See “Environmental considerations” on page xxii.

QSAM allows data areas to be located above the 16 MB line. To take advantage of providing data areas above the 16MB line for QSAM macros, the issuer of the GET macro must then execute in 31-bit addressing mode. To take advantage of QSAM buffers above the line, you must specify for OPEN to obtain the buffers above the line and the issuer of the GET macro must then execute in 31-bit addressing mode. To specify that OPEN is to get buffers above the 16 MB line, code RMODE31=BUFF on the DCBE macro.

Data Conversion: You can request conversion by coding LABEL=(,AL) or (,AUL) in the DD statement, or by coding OPTCD=Q in the DCB macro or DCB subparameter of the DD statement. When conversion is requested, all records whose record format (RECFM parameter) is F, FB, D, DS, DB, DBS, or U are automatically converted from one character representation to another when the input buffer is full. Conversion is performed according to one of the following techniques:

- **Coded Character Set Identifier (CCSID) Conversion.** If CCSIDs are supplied from any source for ISO/ANSI V4 tapes, records are converted from the CCSID which represents the data on tape to the CCSID as seen by the problem program. You can also prevent conversion by supplying a special CCSID. CCSID may be supplied in the CCSID subparameter of a JOB, EXEC, or DD statement or the tape label.
- **Default Character Conversion.** If you are using non-ISO/ANSI V4 tapes or if CCSIDs are not supplied by any source, data management converts the records from ASCII code to EBCDIC code using specific tables defined for this default character conversion.

Refer to *z/OS DFSMS Using Data Sets*, for a complete description of CCSID conversion and default character conversion.

The format of the GET macro is:

[*label*]	GET	{*dcb address*|*pdab address*} [,*area address*] [,TYPE=P]

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for the opened input data set being retrieved.

***pdab address*—RX-Type Address, (2-12), or (1)**
specifies the address of the parallel data access block for the opened input data sets from which a record is retrieved. When *pdab address* is used, TYPE=P must be coded.

***area address*—RX-Type Address, (2-12), or (0)**
specifies the address of an area into which the system is to move the record (move or data mode). The move, locate, or data mode can be used with QSAM, but must not be mixed in the specified data control block. When issued in 31-bit addressing mode, the input area address (move or data mode) must be clean 31-bit addresses. For move or data mode, if the input area address resides above the 16MB line, you must issue the GET in 31-bit mode. If you requested that OPEN get buffers above the 16MB line, the GET must be issued in 31-bit mode. If the *area address* is omitted in the move or data mode, the system assumes that register 0 contains the area address. The following describes the operation of the three modes:

Locate Mode: If locate mode is specified in the data control block, the *area address* must be omitted. The system returns the address of the beginning buffer segment containing the record in register 1. If the data set is open for RDBACK, register 1 points to the last byte of the record. This address remains valid until you issue the next GET, FEOV, RELSE, or CLOSE macro for the DCB. Reasons why the address might become invalid include the system may be reading new data into the old buffer or the system may have freed the buffer.

When retrieving variable-length spanned records, and the logical record interface (LRI) or extended logical record interface (XLRI) is not used, the records are obtained one segment at a time. The problem program must retrieve additional segments by issuing subsequent GET macros, except when a logical record interface is requested (by specifying BFTEK=A in the DCB macro, by issuing a BUILDRCD macro, or by specifying DCBLRECL=0K or nnnnnK in the DCB macro). In this case, the control program retrieves all record segments and assembles the segments into a complete logical record. The system returns the address of this record area in register 1.

When the maximum logical record length is greater than 32756 bytes, LRECL=X must be specified in the data control block, and the problem program must assemble the segments into a complete logical record. LRECL=X or segment mode processing is not allowed for ISO/ANSI spanned records, RECFM=DS or RECFM=DBS.

Move Mode: If move mode is specified in the data control block, the *area address* specifies the beginning address of an area in the problem program into which the system moves the record. If the data set is open for RDBACK, the *area address* specifies the ending address of an area in the problem program.

If move mode is specified in the data control block, do not code BFTEK=A.

For variable-length spanned records, the system constructs the record-descriptor word in the first 4 bytes of the area and assembles one or more segments into the data portion of the logical record area; the segment descriptor words are removed. When XLRI mode is used, the record descriptor word (RDW) in the record area is a fullword value.

Data Mode: If data mode is specified in the data control block (data mode can be specified for variable-length spanned records only), the *area address* specifies the address of the area in the problem program into which the system moves the data portion of the logical record. A record-descriptor word is not constructed when data mode is used. TYPE=P cannot be used with data mode.

Extended Logical Record Interface (XLRI): When the GET macro is used in XLRI mode, the address returned in register 1 points to a fullword record length value. The 3 low-order bytes of the fullword indicate the length of the complete logical record plus 4 bytes for the fullword.

XLRI mode requires a record area to assemble a complete logical record from the segments that are read.

If a record area is not automatically obtained by OPEN processing, you can construct a record by using the BUILDRCD macro before issuing the OPEN. The DCB LRECL field indicates the length of the area in 'K' units (1024 bytes) required to contain the longest logical record of the data set.

Restriction: If spanned records extend across volumes, errors might occur when using the GET macro if a volume that begins with a middle or last record segment is mounted first, or if an FEOV macro is issued followed by a GET macro. QSAM cannot begin reading from the middle of the record. (This applies to move mode, data mode, and locate mode if logical record interface is specified.)

TYPE=P

TYPE=P and *pdab address* are used to retrieve a record from a queue of input data sets that have been opened. The open and close routines add and delete DCB addresses in the queue. The DCB from which a record is retrieved can be located from information in the PDAB. For this purpose, the formatting macro, PDABD, should be used. When *pdab address* is used, TYPE=P must be coded. The TYPE=P parameter is not supported for 31-bit callers. Unpredictable results may occur.

GET routine exits

The end-of-data-set (EODAD) routine is given control if the end of the data set is reached; afterward, the data set must be closed. Issuing a GET macro in the EODAD routine results in abnormal termination of the task.

The error analysis (SYNAD) routine is given control if the input operation could not be completed successfully due to an uncorrectable I/O error. The contents of the general registers when control is given to the SYNAD exit routine are described in “Status information following an input/output operation” on page 373.

If your SYNAD or EODAD routine is entered, it is entered in the addressing mode in which the GET was issued. If you supplied in the DCBE a SYNAD or EODAD routine which resides above the line, then the GET must be issued in 31-bit addressing mode. On entry to the SYNAD routine, register 1 contains error flags in byte 0 followed by the DCB address in bytes 1-3. For 31-bit callers, the caller must save the error flags, if needed, and then clear the high order byte of register 1 before using it to access fields within the DCB in the SYNAD routine.

GETBUF—Obtain a buffer (BDAM, BISAM, BPAM, and BSAM)

The GETBUF macro causes the control program to obtain a buffer from the buffer pool assigned to the specified data control block and to return the address of the buffer in a designated register. The BUFCB

field of the data control block must contain the address of the buffer pool control block when the GETBUF macro is issued. The system returns control to the instruction following the GETBUF macro. Use the FREEBUF macro to return the buffer obtained to the buffer pool. GETBUF does not support buffers above the line.

The GETBUF macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The format of the GETBUF macro is:

[*label*]	GETBUF	*dcb address* ,*register*

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block containing the buffer pool control block address. When issued in 31-bit addressing mode, the input DCB address must be a clean 31-bit address.

***register*—(2-12)**
specifies one of the registers 2 through 12 in which the system places the address of the buffer obtained from the buffer pool. If no buffer is available, the contents of the designated register are set to 0.

GETPOOL—Build a buffer pool (BDAM, BISAM, BPAM, BSAM, QISAM, and QSAM)

The GETPOOL macro builds a buffer pool in a storage area acquired by the system. The system places the address of the buffer pool control block in the BUFCB field of the data control block. If you choose to issue the GETPOOL macro for QSAM and QISAM, then issue it either before an OPEN macro is issued or during the OPEN data control block exit routine for the specified data control block. Otherwise, the system will build an appropriate buffer pool for you. Do not issue the GETPOOL macro if you wish QSAM buffers to be above the 16MB line.

If you choose to issue the GETPOOL macro for BDAM, BISAM, BPAM, or BSAM, then issue it before you issue the GETBUF macro. Remember that if the BUFNO parameter is supplied in the data control block before completion of the OPEN DCB exit routine, then OPEN will build a buffer pool and your program should not issue GETPOOL. You may choose to supply BUFNO when the data set is allocated to the program (on the DD statement) and not clear BUFNO in the DCB before completion of the OPEN DCB exit routine.

The GETPOOL macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

Note: BSAM cannot support 64 bit real storage for any tape devices that do not support 64 bit IDAWs. Applications that use BSAM to process a tape data set can experience ABEND0D3 and/or ABENDB00 due to the inability to use a 64-bit IDAW by the device.

The format of the GETPOOL macro is:

[*label*]	GETPOOL	*dcb address* ,{*number of buffers*,*buffer length*|(0)}

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block to which the buffer pool is assigned. Only one buffer pool can be assigned to a data control block.

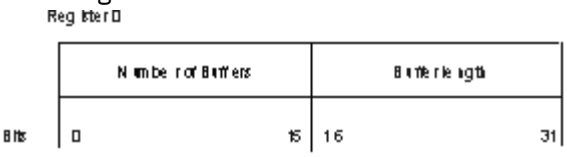
The value you specify can be either a positive or a negative value. If this parameter has the high-order bit on (for example, to signify the last address in a list), this bit must be reset to zero. Otherwise, the address will be treated as a negative value. When issued in 31-bit addressing mode, the input DCB

address must be a clean 31-bit address. The resulting buffer pool always resides below the 16MB line.

***number-of-buffers*—symbol, decimal digit, absexp, or (2-12)**
specifies the number of buffers in the buffer pool to a maximum of 255.

***buffer length*—symbol, decimal digit, absexp, or (2-12)**
specifies the length, in bytes, or each buffer in the buffer pool. The value specified for the buffer length must be a doubleword multiple; otherwise, the system rounds the value specified to the next higher doubleword multiple. The maximum length that can be specified is 32760 bytes. For QSAM, the buffer length must be at least as large as the value specified in the block size (DCBBLKSI) field in the data control block.

(0)
The number of buffers and buffer length can be specified in general register 0. If `(0)` is coded, register 0 must contain the binary values for the number of buffers and buffer length as shown in the following illustration:



Your program releases the buffer pool and the associated storage area by issuing a FREEPOOL macro after issuing a CLOSE macro for the data set indicated in the specified data control block.

IEWLCNVT—Convert directory entries (BPAM)

If your program is accessing both BLDL and DESERV type directory entries, you can use the IEWLCNVT macro to convert one type into the other to provide a single format for processing.

The IEWLCNVT macro provides two functions for directory entry conversion:

- Converting a PDS Directory Entry (PDSDE) to a Program Management Attribute Record (PMAR)
- Converting a PMAR to a PDSDE

Convert a PDSDE to a PMAR

You can convert a PDSDE to a PMAR when you need to convert a directory entry which was obtained from BLDL into PMAR format. When using this macro, you must supply the address of the indicator byte (PDS2INDC) of the PDSDE and an output area of sufficient size for the PMAR to be generated. The length of the PMAR is returned in a full word field supplied by the caller. If the PDSDE is for an alias entry (i.e. the PDS2ALIS bit is on), you must provide an 8-byte area in which the primary name will be returned.

A sufficient size for the PMAR is the length of the PMAR basic section plus the length of the PMARR section.

It is impossible to verify with complete certainty that the input PDSDE is a directory entry for a load module. However, the results of converting a non-load module directory entry into PMAR format would be completely unintelligible. Therefore, IEWLCNVT performs certain minimal checks to ensure that the PDSDE approximates the format of a load module directory entry before processing the conversion. If any of these tests fail, the conversion will not be performed and error return and reason codes will be issued.

Convert a PMAR to a PDSDE

You may use this conversion when converting a directory entry which was obtained from DESERV FUNC=GET, or DESERV FUNC=GET_ALL, into PDSDE format. When invoking this function, the caller supplies the PMAR to be converted and an output area of at least 63 bytes in which the PDSDE will be returned. The input PMAR must include either the PMARR or PMARL extension. You must also specify the FLAGS= parameter that is used to define a byte which indicates processing flags.

The processing flags byte is mapped by the LCNV_FLAGS_DSECT of the IEWLCNV macro. The only processing option defined currently is a bit which indicates whether the input PMAR is for an alias name or not. The PDSDE generated will consist of the indicator byte (PDS2INDC) and the user data field. The fields in the IHAPDS mapping that precede PDS2INDC will not be generated. The length of the PDSDE (which is the length of PDS2INDC, 1 byte, plus the length of the user data) may be returned in a full word field supplied by the caller.

To convert a PMAR for a primary name to a PDSDE, the PMARA parameter should not be specified and the flags parameter should pass a byte of X'00'.

To convert a PMAR for an alias name to a PDSDE, where the PMAR was obtained from DESERV GET or GET=ALL, the PMAR already reflects the attributes for the alias. Therefore, the PMARA parameter should not be specified and the FLAGS parameter should set the LCNV_FLAGS_ALIAS bit to 1.

If this macro is used in the DESERV EXIT routine in response to a DESERV PUT, the input to the exit routine is a single PMAR (for the primary name) and optionally a list of PMARAs (one for each alias name defined). To use this conversion function in this environment to generate a PDSDE for an alias, you must complete the following tasks:

- Pass the PMAR for the primary name via the PMAR parameter.
- Pass the PMARA for the alias via the PMARA parameter.
- Set the LCNV_FLAGS_ALIAS bit and pass this byte via the FLAGS parameter.

To convert PMAR to PDSDE, the format of the IEWLCNVT macro is:

[*label*]	IEWLCNVT	FUNC=PMAR_TO_PDSDE ,FLAGS=*processing_flags* ,PMAR=*pmar_storage* ,PDS2INDC=*pdsde_indicator_byte* [,AMODEREG=*register*] [,PMARA=*pmara_storage*] [,PNAME=*primary_name*] [,MF={S| L| (E,*{(1-12)|RX–type address}* [,COMPLETE|NOCHECK])}] ,OUTLEN=*output_length* [,RETCODE=*retcode*] [,RSNCODE=*rsncode*]

To convert PDSDE to PMAR, the format of the IEWLCNVT macro is:

[*label*]	IEWLCNVT	FUNC=PDSDE_TO_PMAR ,PMAR=*pmar_storage* ,PDS2INDC=*pdsde_indicator_byte* [,PNAME=*primary _name*] [,AMODEREG=*register*] [,MF={S| L| (E,*{(1-12)|label}* [,COMPLETE|NOCHECK])}] ,OUTLEN=*output_length* [,RETCODE=*retcode*] [,RSNCODE=*rsncode*]

AMODEREG=*register*
identifies a register that the macro will use to save and restore the addressing mode of the caller. If the caller is always in 31-bit addressing mode (at the time IEWLCNVT is invoked) you can omit

this parameter. If the caller is in 24-bit addressing mode at the time IEWLCNVT is to be issued, you must specify the AMODEREG parameter. Valid registers are 2-12. The register may be enclosed in parentheses, but this is not required.

FUNC=*function_name*
identifies the function to be performed. The FUNC parameter is not required for MF=L unless other parameters are specified. Valid function_name values are:

PMAR_TO_PDSDE
convert a PMAR to a PDS2 style directory entry.

PDSDE_TO_PMAR
convert the user data of a PDS2 style directory entry (PDSDE) for a load module to a PMAR.

FLAGS=*processing_flags*
specifies options to be used while processing the PMAR_TO_PDSDE function. Variable *processing_flags* is a byte of flags. The only defined flag indicates if an alias entry is being processed. The processing flags byte is mapped by LCNV_FLAGS_DSECT in the IEWLCNV macro.

[,MF={S|L|(E,*{(1-12)|RX-type address}*[,COMPLETE|NOCHECK])}]
first argument — keyword S, L, E or default to S when MF is omitted.

Second argument, if MF=E — registers 1-12 or RX-type address.

Third argument, if MF=E — keyword COMPLETE or NOCHECK or default to COMPLETE, if omitted.

The MF (macro format) keyword specifies how the macro should generate its code. Invalid keyword checking, based on function specified, is done for all macro formats.

The standard format (S) will check all required keywords and invalid keywords. This form generates a complete in-line expansion of the parameter list. Control is then transferred to the convert routine. The standard form is for programs that are not reenterable.

L specifies the list form of the macro. This form generates a remote parameter list. Registers are invalid arguments for MF=L format since executable code generation does not occur, only adcons are generated. Invalid keyword checking is done.

E specifies the execute form of the macro. This form updates the remote parameter list (MF=L) and transfers control to the convert routine. A second parameter is required and a third parameter is optional.

The second parameter for MF=E format is the address of the parameter list created by the MF=L IEWLCNVT invocation. This parameter must be specified as either a RX type of address (possibly the label from MF=L macro invocation) or a register enclosed in parentheses.

The third parameter, COMPLETE or NOCHECK, is optional. Default is COMPLETE. This argument specifies whether required keyword checking will be done.

If NOCHECK is coded, then none, some, or all allowable keywords may be specified. It is assumed that any missing keywords are coded on the MF=L macro invocation. If some keywords are coded, the FUNC keyword is also required to enable keyword validation.

If COMPLETE is coded or allowed to default, the plist will be zeroed out (except for the plist header). All required keywords must be specified.

OUTLEN=*output_length*
specifies a fullword (4-byte) field to contain the length of the generated directory data. Variable *output_length* is an output parameter on the PMAR_TO_PDSDE and PDSDE_TO_PMAR functions. OUTLEN must not be specified on MF=L.

PDS2INDC=*pdsde_indicator_byte*—RX-type address or (2-12) (standard form)
specifies the indicator byte preceding the user data field of a PDS directory entry. Variable *indicator_byte* is an input parameter on the PDSDE_TO_PMAR function and an output parameter on the PMAR_TO_PDSDE function.

PMAR=*pmar_storage*—RX-type address or (2-12) (standard form)
specifies an area mapped by the PMAR structure of macro IEWPMAR. Variable *primary_process_sar_data* is the PMAR structure used for input on the PMAR_TO_PDSDE function and output on the PDSDE_TO_PMAR function.

PMARA=*pmara_storage*—RX-type address or (2-12) (standard form)
specifies an area mapped by the PMARA structure of macro IEWPMAR. Variable *alias_process_sar_data* is the PMARA structure used as input by the PMAR_TO_PDSDE function.

PNAME=*primary_name*—RX-type address or (2-12) (standard form)
specifies the area for an eight byte primary name. This is an input field on the PMAR_TO_PDSDE function and must be passed if the processing flags indicate that an alias is being processed.

RETCODE=*retcode*—RX-type address or (2-12) or (15)
specifies the name of the variable where the macro is to store the return code associated with the result of the function invocation. Variable *return_code* is a fullword value but is optional. If RETCODE is not specified, the return code is in register 15. The RETCODE parameter can not be specified on an MF=L invocation.

RSNCODE=*rsncode*—RX-type address or (2-12) or (0)
specifies the name of the variable where the macro is to store the reason code associated with the result of the function invocation. Variable *reason_code* is a fullword value but is optional. If RSNCODE is not specified, the return code is in register 0. The RSNCODE parameter can not be specified on an MF=L invocation.

IEWLCNVT reason codes

IEWLCNVT reason codes have the following format:

Offset	Length	Meaning
00 (X'00')	1 byte	SMS component code — (X'26') indicates loader (of which IEWLCNVT is a part).
01 (X'01')	1 byte	Module ID— used for problem diagnosis.
02 (X'02')	2 bytes	Reason code that identifies the error. A program testing the IEWLCNVT reason code should only look at this last two bytes. The component id and module id should not be tested. They are reported for diagnostic purposes only.

The following are the two low order byte values for the reason codes which IEWLCNVT may return (sorted by return code).

Return Code	Reason Code	Meaning
00 (X'00')		Successful.
	00 (X'00')	Successful (actually 4 bytes of zeros are set).
16 (X'10')		Caller error.
	50 (X'32')	The level field of the PMAR specified an unsupported level. This is set for PMAR_TO_PDSDE function.

Return Code	Reason Code	Meaning
	24 (X'18')	The input PDSDE does not appear to be that of a load module. This is set for the PDSDE_TO_PMAR function. The problem is one of the following: • The PDS2INDC byte indicated a length less than the minimum for a load module's directory entry user data field. This minimum length is 22 bytes. **Note:** PDS2INDC records the user data length in number of half-words in bits 3 through 7. The minimum number of half-words is 11. • Too many or too few note list TTRs indicated in PDS2INDC. A load module will only have either 1 or 2 note list TTRs. PDS2INDC records the number of note list TTRs in bits 1 and 2.
	26 (X'1A')	The input PDSDE is for a program object. Complete conversion will not be performed.

ISITMGD—Is the data set system-managed? (BPAM, BSAM, QSAM)

The ISITMGD macro allows you to determine certain attributes about the data set being processed. The ISITMGD macro sets some bits in the ISITMGD parameter list which you can test to get information about the data set. You test bits in the parameter list to determine if a data set:

- Is SMS-managed.
- Is a partitioned data set extended (PDSE).
- Is an extended format data set.
- Is a compressed format data set.
- Is an encrypted data set.
- Is a UNIX file.
- Contains data members.
- Contains executable programs.
- Is an unknown data type.

The IGWCISM macro maps the ISITMGD parameter list:

ISMMGD
ON if the data set is system-managed.

ISMPDSE
ON if the data set is a PDSE

ISMDSTRP
ON if the data set is extended format

ISMDCOMP
ON if the data set is a compressed format data set. In this case, ISMDSTRP will also be on.

ISMENCRP
ON if the DASD data set is encrypted by the access methods. In this case, ISMDSTRP or ISMPDSE may also be on.

ISMOMVS
ON if processing a UNIX file

ISMDTREC
ON if the data set is a PDSE record format library containing data members (set only if DATATYPE=YES is specified)

ISMDTPGM
ON if the data set is a PDSE program object library (set only if DATATYPE=YES is specified)

ISMDUNK
ON if the data set is an unknown data type (the data type could not be determined) (set only if DATATYPE=YES is specified)

ISMDSNVER
A one-byte binary field that contains the number of the version of the data set format. A value of 1 or 2 indicates the PDSE version. A value of 0 is for other types of data set. A value of 0 should not be taken to mean that the data set is not a PDSE. One such special case is a partitioned concatenation. In that case this value is 0 even if a PDSE is in the concatenation.

ISMMGENS
ON if the data set is enabled for member generations (ON only for PDSE data set version 2)

You need to supply either the address of the opened DCB or the address of a valid DEB. See "ISITMGD completion codes" on page 281 for the ISITMGD return codes, and *z/OS DFSMS Using Data Sets* for an example of coding the ISITMGD macro.

The ISITMGD macro can be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

R13 must contain the address of an 18 word save area.

The format of the ISITMGD macro is:

[*label*]	ISITMGD	{DEB=*addr*|DCB=*addr*} [,CONCAT={**0**|*number*|ALL}] [,DATATYPE={YES|**NO**}] [,MF=S] [,VERSION={**1**|2}]

DEB=*addr*
specifies the address of a valid data extent block.

***addr*—A-type address, or (2-12)**
specifies an in-storage address of the DEB.

DCB=*addr*
specifies the address of a DCB opened to a data set.

***addr*—A-type address, or (2-12)**
specifies an in-storage address of the opened DCB.

CONCAT={0|*number*}
specifies the concatenation number of the PDSE or partitioned data set. This is supported for BPAM and ignored for BSAM and QSAM. For a sequential concatenation ISITMGD always tests the current data set.

0
Indicates the only data set or the first data set in the concatenation.

number
Indicates which data set in the concatenation.

ALL
Indicates the status of all of the data sets in the concatenation. If specified for a sequential concatenation (DSORG=PS), information is returned for only the data set currently positioned to. If specified for a partitioned concatenation (DSORG=PO), information is returned for all the data sets in the concatenation. ISMDSALL in ISMOFLG2 is on if all data sets in the concatenation are of the same type (all partitioned data sets, all PDSEs, all SMS, or all non-SMS).

An application which may run on a release of DFP prior to DFSMS/MVS 1.1.0 and makes use of CONCAT=ALL must determine if CONCAT=ALL was recognized by the ISITMGD execution module at run time. If CONCAT=ALL is recognized, at least two of the data set organization bits of the ISMOFLG2 byte will be set on. If CONCAT=ALL is not recognized, then information returned will be for the first data set in the concatenation (equivalent to CONCAT=0).

DATATYPE={YES|NO}
specifies information on the data type.

YES
Returns information on the data type of the specified data set. ISITMGD can only determine data type information for PDSEs with existing members.

An application that can release of DFP prior to DFSMS/MVS 1.1.0 and makes use of DATATYPE=YES must determine if DATATYPE=YES was recognized by the ISITMGD execution module at run time. If DATATYPE=YES is recognized, at least one of the data type bits in ISMOFLG3 will be set on. If DATATYPE=YES is not recognized, all of the data type bits in ISMOFLG3 will be set off.

The data type information returned, in byte ISMOFLG3, will indicate one or more of the following conditions:

- ISMDTPGM—data type is PDSE program object library.
- ISMDTREC—data type is PDSE record format members (data members).
- ISMDTUNK—data type is unknown. An indication of unknown might indicate that the data set is either a sequential data set, a partitioned data set, or a PDSE with no existing members. A PDSE with no members would have an unknown data type, ISMDTUNK=ON, but would have a data set organization of PDSE, ISMPDSE=ON.
- ISMDSTRP in ISMOFLG2—data set is extended format.

NO
Data type is not determined.

If DATATYPE=NO is specified, or defaulted, no attempt will be made to determine the data type. DATATYPE=NO should be specified explicitly or by default unless the application requires the data type organization, since there is significant additional overhead required to obtain the data type information.

MF=S
specifies the standard form of ISITMGD.

VERSION={1|2}
specifies the version of the parameter list as mapped by IGWCISM.

1
Requests the original 28-byte parameter list. This is the default.

2
Requests the 40-byte parameter list.

The new VERSION keyword can be specified on the standard, list and execute forms of the ISITMGD macro.

Use VERSION=2 to obtain the 40-byte parameter list which will contain the following new field to be set by the system and can be tested after the DCB is open.

ISMBPRFX
A one-byte binary field that contains the length of the prefix associated with each physical block in the data set.

- For encrypted basic and large format sequential data sets, a value of 8 will be returned. However, if the encrypted data set is created without block prefixes (VVREDSENCNP = on), a value of 0 will be returned.
- For other data set types, a value of 0 will be returned.

ISITMGD—List form

The list form of the ISITMGD macro is used to construct a parameter list for the ISITMGD function. The IGWCISM macro maps the ISITMGD parameter list.

The list form of the ISITMGD macro is shown below. The description of the standard form of the ISITMGD macro explains the function of each parameter.

[*label*]	ISITMGD	{DEB=*addr*|DCB=*addr*} [,CONCAT={0|*number*|ALL}] [,DATATYPE={YES|NO}] ,MF=L

DEB=*addr*–RX-type address or (2–12)

DCB=*addr*–RX-type address or (2–12)

CONCAT={0|*number*|*ALL*}

DATATYPE={YES|NO}

MF=L
specifies the list form of ISITMGD. This generates a parameter list that contains no executable instructions. This parameter list is mapped by the IGWCISM macro. The list can be used as input to and be modified by the execute form.

ISITMGD—Execute form

A remote parameter list is used in, and can be modified by, the execute form of the ISITMGD macro.

Rule: If either the DATATYPE keyword (DATATYPE=YES or DATATYPE=NO) or CONCAT=ALL is specified, your application program is responsible for initializing the parameter list. You can initialize the parameter list simply by invoking the ISITMGD MF(L) format, which will result in an initialized static parameter list. If dynamic storage is obtained for the parameter list, it must be initialized by copying a static parameter list.

The execute form of the ISITMGD macro is shown below. The description of the standard form of the ISITMGD macro explains the function of each parameter.

[*label*]	ISITMGD	{DEB=*addr*|DCB=*addr*} [,CONCAT={0|*number*|ALL}] [,DATATYPE={YES|NO}] ,MF=(E,*list_address*)

DEB=*addr*–RX-type address or (2–12)

DCB=*addr*–RX-type address or (2–12)

CONCAT={0|*number*|*ALL*}

DATATYPE={YES|NO}

MF=(E,*list_address*)
specifies the execute form of ISITMGD, and an existing parameter list is used.

***list_address*—RX-Type address or (2-12)**
specifies the address of the parameter list.

ISITMGD completion codes

When the system returns control to your problem program, the low-order byte of register 15 contains a return code. The low-order byte of register 0 contains a reason code.

The ISITMGD return and reason codes are as follows:

Return Code (15)	Reason Code (0)	Meaning
00 (X'00')		Successful completion.
04 (X'04')	04 (X'04')	Data control block was not open.

Return Code (15)	Reason Code (0)	Meaning
04 (X'04')	08 (X'08')	Data extent block was not valid.
04 (X'04')	16 (X'10')	An access method control block (ACB), not a DCB, was supplied.
04 (X'04')	20 (X'14')	DEB extension does not exist.
04 (X'04')	28 (X'1C')	Access method type is not supported.
04 (X'04')	32 (X'20')	Invalid unit control block (UCB).
04 (X'04')	36 (X'24')	Invalid SMS control block.
04 (X'04')	40 (X'28')	Invalid SMS control block. This could be caused by a bad DCB, a DEB error, or an internal SMS error.
04 (X'04')	48 (X'30')	Invalid INOUT DCB or DEB address. The input DCB and DEB control blocks did not point to each other.
04 (X'04')	52 (X'34')	DEBCHK error.
08 (X'08')	00 (X'00')	ISITMGD macro is not supported on current level of system. Must be MVS/DFP 3.2 or later.
12 (X'0C')	00 (X'00')	Reserve bits in the parameter list are set on, possibly function requested which is not supported on this level of ISITMGD.
12 (X'0C')	04 (X'04')	Invalid parameter list pointer.
12 (X'0C')	08 (X'08')	Invalid parameter list level.
12 (X'0C')	12 (X'0C')	Invalid parameter list length.
12 (X'0C')	16 (X'10')	Invalid concatenation number.
12 (X'0C')	20 (X'14')	Invalid concatenation number.
12 (X'0C')	24 (X'18')	DCB or DEB pointer is zero.
12 (X'0C')	28 (X'1C')	The bits indicating the DCB and DEB are either both on or off. Either both the DCB and DEB were supplied or neither.
16 (X'10')	04 (X'04')	Data type not set due to SMS error, dump taken.
16 (X'10')	08 (X'08')	Data type not set due to SMS error, no dump taken.

MSGDISP—Displaying a ready message (BSAM, QSAM)

The MSGDISP macro is used to load the message display on magnetic tape drives that use cartridges, such as the 3480, 3490 and 3590. Functions for the display include:

- Displaying a ready message
- Mount volume
- Demount volume
- Reset display
- Verify volume
- Generalized display

These MSGDISP macro functions are explained in *z/OS DFSMSdfp Advanced Services*.

The MSGDISP macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The standard form of the MSGDISP macro is:

[*label*]	MSGDISP	RDY ,DCB=*addr* [,TXT={'*msgtxt*'|*addr*}]

RDY
specifies that text supplied in the TXT parameter is displayed in positions 2 through 7 of the display while the data set is open. The display is steady (not flashing) and is enclosed in parentheses. The display is also written to the tape pool console (routing code 3, descriptor code 7).

DCB=*addr*–A-type address, or (2-12)
specifies the address of a DCB that is opened to a data set on the mounted volume. If multiple devices are allocated, the message display is directed to the one containing the volume currently in use.

Recommendation: If multiple devices or multiple volumes are allocated, you can update a message display after an end-of-volume condition by using the EOV exit specified in a DCB exit list. For a concatenated data set with unlike characteristics, you can also use the DCB open exit to update the display.

TXT={'*msgtxt*'|*addr*}
specifies as many as 6 characters to be displayed in positions 2 through 7. If TXT is not specified, blanks are displayed.

'*msgtxt*'
specifies the 1- to 6-character text. The text must be enclosed in apostrophes.

***addr*—A-type address, or (2-12)**
specifies an in-storage address of an area containing the six bytes of text to be displayed.

MSGDISP—List form

The list form of the MSGDISP macro is:

[*label*]	MSGDISP	[RDY] [,DCB=*addr*] ,MF=L [,TXT={'*msgtxt*'|*addr*}]

MF=L
specifies the list form of MSGDISP. This generates a parameter list that contains no executable instructions. The list can be used as input to and can be modified by the execute form.

TXT={'*msgtxt*'|*addr*}

'*msgtxt*'
specifies the 1- to 6-character text. The text must be enclosed in apostrophes.

***addr*—A-Type address**
specifies an in-storage address of an area containing the text to be displayed.

MSGDISP—Execute form

The execute form of the MSGDISP macro is:

[*label*]	MSGDISP	RDY [,DCB=*addr*] ,MF=(E,*addr*) [,TXT={'*msgtxt*'|*addr*}]

RDY

DCB=*addr*—RX-Type address or (2-12)

MF=(E,*addr*)
specifies that the execute form of MSGDISP and an existing parameter list is to be used.

***addr*—RX-Type address, (1), or (2-12)**
specifies an in-storage address of the parameter list.

TXT={'*msgtxt*'|*addr*}

'*msgtxt*'
specifies the 1- to 6-character text. The text must be enclosed in apostrophes.

***addr*—RX-Type address or (2-12)**
specifies an in-storage address of an area containing the six bytes of text to be displayed.

MSGDISP completion codes

When the system returns control to your problem program, the low-order byte of register 15 contains a return code. For return code = 08, the low-order byte of register 0 contains a reason code.

The MSGDISP return and reason codes are:

Return Code (15)	Reason Code (0)	Meaning
00 (X'00')		Successful completion.
04 (X'04')		Device does not support MSGDISP.
08 (X'08')	01 (X'01')	Invalid parameter.
08 (X'08')	02 (X'02')	Invalid DCB or DEB error.
08 (X'08')	03 (X'03')	Environmental error.
08 (X'08')	04 (X'04')	Authorization violation.
08 (X'08')	05 (X'05')	Invalid UCB.
08 (X'08')	06 (X'06')	Invalid request.
08 (X'08')	11 (X'0B')	Unsuccessful ESTAE macro call.
08 (X'08')	12 (X'0C')	Insufficient virtual storage available.
12 (X'0C')		I/O error. **Note:** An I/O error occurs for load display if the drive display has a hardware failure.

NOTE—Provide relative position (BPAM and BSAM—tape and DASD only)

The NOTE macro returns the position of the last (or next if TYPE=ABS is specified) block read from or written into a data set. All input and output operations using the same data control block must be tested for completion before the NOTE macro is issued.

The NOTE macro with the REL parameter, which is the default, works with any magnetic tape drive. However, NOTE with the ABS parameter works only on cartridge tapes, such as the 3480, 3490 and 3590.

The capability of using the NOTE macro is automatically provided when a PDSE or partitioned data set is used (DSORG=PO). But you must specify MACRF=P in the DCB macro to use NOTE or POINT when using BSAM for a sequential data set, a large format data set, or a member of a partitioned data set or PDSE.

The NOTE macro can be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

Spooled Data Sets: The NOTE (and POINT) macros cannot be used with spooled data sets.

Subsystem data sets: The NOTE macro can be used for subsystem data sets if the subsystem supports it. If the subsystem does not support it, the results are unpredictable.

UNIX files: The NOTE macro can be issued for UNIX files, except for FIFO or character special files or with PATHOPTS=OAPPEND.

If you set BLOCKTOKENSIZE=LARGE (DCBELARGE=1) and MACRF and DSORG in the DCB indicate BSAM and the NOTE and POINT macros then the OPEN macro will allow the opening of large format data sets. BLOCKTOKENSIZE=LARGE signifies that you aware of the changes in the DSCB, DEB, and TTR conversion routines and that the NOTE and POINT macros must use the TTTR interface in place of the TTR0 interface regardless of what kind of DASD the data set resides on.

When you code BLOCKTOKENSIZE=LARGE on the DCBE with BSAM or BPAM then:

- For a sequential non-extended format data set or PDS, NOTE and POINT will use TTTR rather then TTRx. In a PDS the high order byte will always be 0. There will be no effect on the FIND macro which will continue to accept a four byte TTRc where c is the concatenation number as received from BLDL. The content of DCBRELAD and DCBERELA, which identify the location of the beginning of the member, will remain unchanged.
- For extended format, compressed format, or UNIX files, the relative block number will be expressed in four bytes rather then the three high order bytes. This removes a size restriction for the NOTE and POINT macros.
- For a PDSE member, you will receive a four byte simulated TTTR value instead of the previous three byte field. The maximum number of records will increase from 15,728,639 to 4,277,145,599.

  Note: If you pass to POINT a TTR returned from BLDL you must right align the three bytes returned into the four byte field that you pass.

The format of the NOTE macro is:

[*label*]	NOTE	*dcb address* [,TYPE={ABS|<u>REL</u>}]

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block opened for the partitioned or sequential data set being processed. For TYPE=REL requests, when issued in 31-bit addressing mode, the input DCB address must be a clean 31-bit address.

TYPE={ABS|<u>REL</u>}
indicates if the device that the data set resides on supports the physical block identifier (ABS) or relative addresses (REL). The ABS option of the POINT macro generally is much faster than the REL option, especially when positioning over many blocks.

ABS
specifies that, after NOTE executes successfully (contents of register 15 is 0), register 0 contains the physical block identifier for the next data block that will be transferred between virtual storage and the control unit buffer, and register 1 contains the physical block identifier of the next data block that will be transferred between the control unit buffer and the tape drive.

You can subtract register 1 from register 0. On certain cartridge tape drives the physical block identifier is a 32-bit unsigned integer. In these cases the result is the number of data blocks left in the control unit buffer. On other drives the block identifier contains unsigned integers of unequal length. The number of bits varies between models. Consult the hardware manual. A negative result means the buffer is in read mode, and a positive remainder means the buffer is in either write or read-backward mode. A zero remainder means that no data is buffered.

REL

causes the system to return the relative position of the last block read from or written into a data set. This means that if the data set later is copied and the number of blocks on each volume is changed or the data set is reblocked, then the value returned by NOTE for a particular block may differ. The position of the current volume is returned in register 1 as follows:

Magnetic Tape: The block number is in binary, right-adjusted in register 1 with high-order bits set to zero. Do not use a NOTE macro for tapes without standard labels when:

- The data set is opened for RDBACK (specified in the OPEN macro), *or*
- The DISP parameter of the DD statement for the data set specifies DISP=MOD and the OPEN option was OUTPUT or OUTIN, *or*
- The OPEN option was EXTEND or OUTINX, *or*
- The CLOSE macro is issued with TYPE=T and the LEAVE option, *or*
- The CLOSE macro is issued with TYPE=T and the DISP option and the DD statement or dynamic allocation equivalent has DISP=(xxx,PASS).

Direct Access Storage Devices: The value that the NOTE macro returns depends on the type of data set and on whether you coded BLOCKTOKENSIZE=LARGE on the DCBE macro that the DCB points to. If your program turns on the DCBELARGE bit in the DCBE, it has the same effect as coding BLOCKTOKENSIZE=LARGE.

If you do not code BLOCKTOKENSIZE=LARGE, then the value returned in register 1 is in the three high order bytes and the low order byte contains X'00'. If you code BLOCKTOKENSIZE=LARGE, then the value returned in register 1 is in all four bytes. BLOCKTOKENSIZE=LARGE has no effect on the FIND and BLDL macros or on the content of DCBRELAD and DCBERELA, which identify the location of the beginning of the member. They always use a three-byte member token.

If your program runs on a system before z/OS 1.7, then the system ignores any setting of BLOCKTOKENSIZE=LARGE.

TTRz Basic format, large format. or PDS:

TTRz
Returned if you do not code BLOCKTOKENSIZE=LARGE.

TTTR
Returned if you code BLOCKTOKENSIZE=LARGE. You must code this if the data set is large format unless less then 65536 tracks are allocated to it on the volume and your program opened the data set with the INPUT option. If the data set is a PDS or basic format data set, then the high order byte always will contain X'00' because those data sets cannot contain more than X'FFFF' tracks.

Note: The TT or TTT bytes contain the relative track address of the block, where the first track in the data set is 0.

R
Specifies the number of the block on the track, where the first block is X'01'.

Z
Contains X'00'.

- PDSE or UNIX file with BPAM: The three or four byte token does not represent the physical location of the data set or member. Without BLOCKTOKENSIZE=LARGE, the maximum number of records in a member is 15,728,639. With BLOCKTOKENSIZE=LARGE, the maximum number of records in a member is 4,277,145,599.
- UNIX file with BSAM: The NOTE macro can be issued for UNIX files, except for FIFO or character special files or with PATHOPTS=OAPPEND. If you do not code BLOCKTOKENSIZE=LARGE, then (1) NOTE does not support a UNIX file that contains more than 16 mega-records minus two (X'1000000'-2 or 16,777,214) and (2) a NOTE macro after 16 mega-records minus two returns an invalid value (X'FFFFFF'). If you code BLOCKTOKENSIZE=LARGE, then you do not have this limit.

- Extended format data set or compressed format data set: The three or four byte value is a token that does not represent the physical location of the data set or member. If you do not code BLOCKTOKENSIZE=LARGE, then your program can address up to 16,777,215 blocks. If you code BLOCKTOKENSIZE=LARGE, then your program can address up to 4,294,967,295 blocks.

If the data set later is copied to a DASD that has a different track length, the value returned by NOTE for a particular block may differ.

When direct access storage devices are being used, the amount of remaining space on the track is returned in register 0 if a NOTE macro follows a WRITE macro. If a NOTE macro follows a READ or POINT macro, the track capacity of the direct access storage device is returned in register 0. For PDSEs, extended format data sets, and UNIX files, the NOTE macro does not calculate the amount of space remaining on the track or the track capacity, and returns a value of X'7FFF'.

Recommendation: Your programs should not become device-dependent. Your program is device-dependent if it examines what NOTE returns in register 1 or performs arithmetic on it. Your program can pass the four bytes to the POINT macro without examining them.

An example of an unmovable data set is one that has all of these attributes:

- The system determined the block size because it was omitted. IBM recommends omitting it. See the BLKSIZE parameter description for "DCB—Construct a data control block (BSAM)" on page 189.
- The data set resides on DASD. A program such as DFSMShsm may copy the data set to a different type of DASD or to tape. This may cause the system to determine a different block size that is optimized for the new device type.
- A program has stored the results of a NOTE macro inside the data set or in some other data set. This value typically depends on the block size.

NOTE completion codes

The ABS parameter causes NOTE to return a return (completion) code. The REL parameter does not cause a return code.

If TYPE=ABS is specified

When the system returns control to your program and you have specified the ABS parameter, the low-order byte of register 15 contains a return code. If return code = 08, the low-order byte of register 0 contains a reason code. The NOTE return and reason codes are:

Return Code (15)	Reason Code (0)	Meaning
00 (X'00')		Successful completion.
04 (X'04')		Device does not support block identifier.
08 (X'08')	01 (X'01')	Incorrect parameter.
08 (X'08')	02 (X'02')	Incorrect DCB or a DEB error.
08 (X'08')	03 (X'03')	Environmental error.
08 (X'08')	11 (X'0B')	Unsuccessful call to ESTAE macro.
08 (X'08')	12 (X'0C')	Insufficient virtual storage available.
12 (X'0C')		Input/output error.

If TYPE=REL is specified

None.

OPEN—Connect program and data (BDAM, BISAM interface to VSAM, BPAM, BSAM, QISAM interface to VSAM, and QSAM)

The OPEN macro completes the specified data control blocks and prepares for processing the data sets identified in the data control blocks. Input labels are analyzed and output labels are created. Control is given to exit routines as specified in the data control block exit list. The processing method (option 1) provides volume positioning for the data set and define the processing mode (INPUT, OUTPUT, and so forth) for the data sets. Final volume positioning (when volume switching occurs) can be specified (option 2) to override the positioning implied by the DD statement DISP parameter. Option 2 applies only to volumes in a multivolume tape data set other than the last volume. Any number of data control block addresses and associated options can be specified in the OPEN macro.

You must supply a DD statement or the dynamic allocation equivalent. The amount of information in the DD statement is up to you, but you must specify the device allocation and a DDNAME that corresponds to the DDNAME keyword of the associated data control block.

All DCBs must reside below the 16 MB line. The real address can be below or above the 16 MB line or above the 2 GB bar.

The OPEN macro does not support more than a total of 255 spooled, SUBSYS or compressed format data sets for one invocation.

The following DCB access methods support the NOCAPTURE option of dynamic allocation: BPAM, BSAM, and QSAM. The following DCB access method does not support the NOCAPTURE option of dynamic allocation: BDAM.

If associated data sets for a 3525 card punch are being opened, all associated data sets must be open before an I/O operation is initiated for any of the data sets.

Note: You can put more than one ACB/DCB in an OPEN/CLOSE macro or you can include BOTH ACBs and DCBs. If multiple ACBs/DCBs are provided, data areas associated with each entry will not be available for reference until the entire OPEN/CLOSE is complete.

HFS data sets: Open and end-of-volume processing do not support HFS data sets. If an application attempts to open a DCB for an HFS data set, the system issues an information message and the current task abends. If an application encounters an end-of-volume condition which causes positioning to an HFS data set, the system issues an information message and the task abends.

HFS and UNIX files: The OPEN macro supports UNIX files. One type of UNIX file is HFS file. OPEN allows you to specify PATH= on a DD statement for a DCB with DSORG=PS and a BSAM or QSAM MACRF. See *z/OS DFSMS Using Data Sets* for more information on this type of access.

Large Format Data Sets: On systems at z/OS V1R7 and above, if you code BLOCKTOKENSIZE=LARGE on the DCBE macro it indicates that your program supports large format data sets. OPEN will also allow you to open large format data sets for input without the BLOCKTOKENSIZE=LARGE if the data contains no more than 65535 tracks. This allows for programs which have not been modified to support large format data sets to be compatible as long as the data sets are not too large.

The standard form of the OPEN macro is as follows (the list and execute forms are shown following the description of the standard form):

[*label*]	OPEN	(*dcb address*[, [(*options*)][,...]]) [,TYPE=J] [,MODE=24|31]

***dcb address*—A-type address or (2-12)**
specifies the address of the data control blocks for the data sets to be prepared for processing.

If the register format is specified, then the register must be enclosed within parentheses. For example, OPEN ((2),INPUT).

options

The option values shown in the following table indicate the volume positioning available based on the device type and access method being used.

Access Method	Device Type					
	Magnetic Tape		Direct Access Storage Device or TSO terminal		Other Types	
	Option 1	**Option 2**	**Option 1**	**Option 2**	**Option 1**	**Option 2**
QSAM	[INPUT] [EXTEND] [OUTPUT] [RDBACK]	[,REREAD] [,LEAVE] [,DISP]	[INPUT] [EXTEND] [OUTPUT] [UPDAT]	—	[INPUT] [EXTEND] [OUTPUT]	—
BSAM	[INPUT] [EXTEND] [OUTINX] [OUTPUT] [INOUT] [OUTIN] [RDBACK]	[,REREAD] [,LEAVE] [,DISP]	[INPUT] [EXTEND] [OUTINX] [OUTPUT] [INOUT] [OUTIN] [UPDAT]	—	[INPUT] [OUTPUT]	—
QISAM Load Mode	—	—	[OUTPUT] [EXTEND]	—	—	—
BPAM	—	—	[INPUT] [INPUT] [OUTPUT] [UPDAT]	—	—	—
BDAM	—	—	[INPUT] [INPUT] [OUTPUT]	—	—	—

If option 1 is omitted, INPUT is assumed. If option 2 is omitted, DISP is assumed. You must code option 1 if also coding option 2. Option 2 has an effect only for multivolume tape data sets. Options 1 and 2 are ignored for BISAM interface to VSAM and QISAM interface to VSAM (in the scan mode), and the data control block indicates the operation. You must specify OUTPUT, OUTIN, OUTINX (DASD), or EXTEND (on DASD) when creating a data set.

Restrictions are as follows:

- The EXTEND, INOUT, OUTIN, and OUTINX options are not allowed for ISO/ANSI Version 3 tape processing. This restriction does not apply to ISO/ANSI Version 4 IBM formatted tapes.
- The UPDAT option is not allowed for compressed format data sets.
- TSO terminals support only QSAM and BSAM

The following describes the options shown in the preceding illustration. All option parameters are coded as shown.

Option 1

Meaning

EXTEND

The data set is treated as an OUTPUT data set, except that records are added to the end of the data set regardless of what was specified on the DISP parameter of the DD statement.

INPUT
Input data set.

INOUT
The data set is first used for input and, without reopening, is used as an output data set. The data set is processed as INPUT if it is a SYSIN data set, or a PDSE, or LABEL=(,,,IN) is specified in the DD statement.

OUTPUT
Output data set (for BDAM, OUTPUT is equivalent to UPDAT).

OUTIN
The data set is first used for output and, without reopening, is used as an input data set. The data set is processed as OUTPUT if it is a SYSOUT data set, or a PDSE, or LABEL=(,,,OUT) is specified in the DD statement.

OUTINX
The data set is treated as an OUTIN data set, except that records are added to the end of the data set regardless of what was specified on the DISP parameter of the DD statement. For PDSEs, OUTINX is equivalent to OUTPUT.

RDBACK
Input data set, positioned to read backward.

Restriction: Variable-length records cannot be read backward. The RDBACK option is not allowed for DASD data sets.

UPDAT
Data set to be updated in place or, for BDAM, blocks are to be updated or added. If you specify UPDAT using QSAM, you must use locate mode.

Restriction: The UPDAT option is not allowed for compressed format data sets, or UNIX files, or for magnetic tapes, or with large block interface.

Option 2
Meaning

LEAVE
Positions the current tape volume to the logical end of the data set when volume switching occurs. If processing was forward, the volume is positioned to the end of the data set. If processing was backward (RDBACK), the volume is positioned to the beginning of the data set.

REREAD
Positions the current tape volume to reprocess the data set when volume switching occurs. If processing was forward, the volume is positioned to the beginning of the data set. If processing was backward (RDBACK), the volume is positioned to the end of the data set.

DISP
Specifies that a tape volume is disposed of in the manner implied by the DD statement associated with the data set. Direct access volume positioning and disposition are not affected by this parameter of the OPEN macro. There are several dispositions that you can specify in the DISP parameter of the DD statement. DISP can be PASS, DELETE, KEEP, CATLG, or UNCATLG. This option has significance at the time an end-of-volume condition is found only when DISP is PASS. The end-of-volume condition might result from issuing an FEOV macro or might be the result of reaching the end of a volume.

If DISP is PASS in the DD statement, the tape is spaced forward to the end of the data set on the current volume.

If any DISP option is coded in the DD statement (except when DISP is PASS), the resultant action when an end-of-volume condition arises depends on (1) how many tape units are allocated to the data set and (2) how many volumes are specified for the data set in the DD statement. This is determined by the UNIT and VOLUME parameters of the DD statement associated with the data set. If the number of volumes is greater than the number of units allocated, the current volume is rewound and unloaded. If the number of volumes is less than or equal to the number of units, the current volume is merely rewound.

Note: When the DELETE option is specified, the system waits for the completion of the rewind operation before it continues processing subsequent reels of tape.

If you code DISP and issue a CLOSE TYPE=T, LEAVE processing is performed. Any other options specified for CLOSE TYPE=T besides LEAVE and REREAD are treated as LEAVE during execution.

TYPE=J
You can code OPEN TYPE=J to specify that, for each data control block referred to, you have supplied a job file control block (JFCB) for use during initialization. A JFCB is an internal representation of information in a DD statement or dynamic allocation. This option, because it may be used with modifying a JFCB, should be used only by the system programmer or only under the system programmer's supervision. MODE=31 is not allowed when TYPE=J is specified. Data sets allocated with the insulated DD attribute are not supported with TYPE=J. For more detailed information on using TYPE=J, see *z/OS DFSMSdfp Advanced Services*.

UNIX files: When you specify TYPE=J, you cannot add or change the value of PATH=. Only changes to LRECL, BLKSIZE, RECFM, BUFNO, and NCP have an effect.

MODE=24|31
You can code OPEN MODE=31 to specify a long form parameter list that can contain 31-bit addresses. Your program does not need to be executing in 31-bit addressing mode to use MODE=31 in the OPEN macro. This parameter specifies the form of the parameter list, not the addressing mode of the program. The default, MODE=24, specifies a short form parameter list with 24-bit addresses. MODE=31 is not permitted if TYPE=J is specified. If TYPE=J is specified, you must use the short form parameter list.

The caller of the standard form of the macro with the short form of the parameter list must reside below the 16 MB line, but the caller can be executing in 31-bit mode. If you code the short form, all ACBs and DCBs must reside below the 16MB line.

The long form parameter list can reside above or below the 16MB line. Although the ACB or DCB address is contained in a 4-byte field, the DCB must be below the 16MB line. Except for VSAM or VTAM ACBs, all ACBs must also be below the 16MB line. Therefore, the leading byte of the ACB or DCB address must contain zeros. If the byte contains something other than zeros, an IEC190I message is issued and the data set is not opened. The program is not abnormally terminated unless an attempt is made to read to or write from the data set.

The following errors in opening a DCB cause the results indicated:

Attempting to open a data control block that is already open
No action

Attempting to open a data control block when the *DCB address* does not specify the address of a data control block.
Unpredictable

Attempting to open a data control block when a corresponding DD statement has not been provided.
A "DD STATEMENT MISSING" message is issued. An attempt to use the data set causes unpredictable results (see note “1” on page 291).

Notes:

1. You can test bit 3 of the DCBOFLGS field in the data control block. Bit 3 is set to 1 if the data control block opened successfully, but is set to 0 if an error occurs, and can be tested by the sequence:

```
TM DCBOFLGS,DCBOFOPN
BZ ERRORRTN       (Branch to your error routine)
```

   If OPEN gives return code 0, then bit 3 for each DCB is 1.

   Executing the two instructions shown above requires writing a DCBD macro in the program, and a base register must be defined with a USING statement before the instructions are executed.
2. Other errors detected by OPEN result in an abend with a system completion code in the form x13, where x is a hex digit from 0 to F. See *z/OS MVS System Codes* for the abend codes.

OPEN return codes

When your program receives control after issuing an OPEN macro, the return code in register 15 indicates if all the data sets were opened successfully. OPEN can fail with a non-zero return code and a message, but no ABEND.

The OPEN return codes are:

Return Code (15)	Meaning
0(X'0')	All data sets were opened successfully and DCBOFOPN (bit 3 of DCBOFLGS) for each DCB is 1. **Note:** If the application has a DCB ABEND EXIT, and that exit ignores an open determinate abend, then open will get a return code=0 even though the dcb is not open. the application should check DCBOFOPN to see if the dcb really is open in that case.
4(X'4')	All data sets were opened successfully, but one or more attention messages were issued.
8(X'8')	At least one data set (VSAM or non-VSAM) was not opened successfully; the ACB or DCB was restored to the contents it had before OPEN was issued; or, if the data set was already open, the ACB or DCB remains open and usable and is not changed.
12(X'C')	A non-VSAM data set was not opened successfully when a non-VSAM and a VSAM data set were being opened at the same time; the non-VSAM data control block was not restored to the contents it had before OPEN was issued (and the data set cannot be opened without restoring the control block).
16(X'10')	One or more of the access method control blocks (ACBs) specified the RLS option but the system has not been set up for RLS (the SMSVSAM server address space is not available). For other DCBs and ACBs any condition described by other return codes is possible.

OPEN—List form

The list form of the OPEN macro constructs a data management parameter list. You can specify any number of parameters (DCB addresses and associated options).

There are two forms of the list, the short form and the long form. The short form list consists of a one-word entry for each DCB or ACB in the parameter list. The high-order byte is used for the options and the 3 low-order bytes are used for the DCB address. The long form list consists of an eight byte entry for each DCB or ACB in the parameter list. The high order byte is used for the options and the low order four bytes are used for the DCB or ACB address. For either form of list, the end of the list is indicated by a 1 in the high-order bit of the last entry's option byte. The length of a list generated by a list form instruction must be equal to the maximum length list required by any execute form instruction that refers to the same list. A maximum length list can be constructed by one of two methods:

- Code a list-form instruction with the maximum number of parameters required by an execute form instruction that refers to the list.
- Code a maximum length list by using commas in a list-form instruction to acquire a list of the appropriate size. For example, coding OPEN (,,,,,,,,),MF=L would provide a list of 5 fullwords (5 DCB addresses and 5 options).

Entries at the end of the list not referred to by the execute-form instruction are assumed to have been filled in when the list was constructed or by a previous execute-form instruction. Before using the execute-form instruction, you can shorten the list by placing a 1 in the high-order bit of the last DCB entry to be processed.

A zeroed work area on a fullword boundary is equivalent to OPEN (,(INPUT,DISP),...),MF=L and can be used in place of a list-form instruction. Allocate four bytes per entry if you wish the effect of MODE=24. Allocate eight bytes per entry if you wish the effect of MODE=31. The high-order bit of the last DCB entry must contain a 1 before this list can be used with the execute-form instruction.

A parameter list constructed by an OPEN, list-form, macro can be referred to by either an OPEN or CLOSE execute form instruction. The description of the standard form of the OPEN macro explains the function of each parameter.

The list form of the OPEN macro is:

[*label*]	OPEN	([*dcb address*],[(*options*)],...) ,MF=L [,TYPE=J] [,MODE=24|31]

dcb address—A-Type Address

MF=L
specifies that the OPEN macro is used to create a data management parameter list that is referred to by an execute form instruction.

TYPE=J
coded the same as the standard form. This has no effect on the macro expansion.

MODE=24|31
coded the same as the standard form. This specification must match that of the execute form. Errors and unpredictable results occur if the modes are inconsistent.

OPEN—Execute form

A remote data management parameter list is used in, and can be modified by, the execute form of the OPEN macro. The parameter list can be generated by the list form of either an OPEN or CLOSE macro.

The description of the standard form of the OPEN macro explains the function of each parameter. The execute form of the OPEN macro is:

[*label*]	OPEN	[([*dcb address*],[(*options*)],...)] ,MF=(E,*data management list address*) [,TYPE=J] [,MODE=24|31]

dcb address—RX-Type Address or (2-12)

MF=(E,*data management list address*)
specifies that the execute form of the OPEN macro is used, and that an existing data management parameter list (created by a list-form instruction) is used. MF= is coded as follows:

E

data management list address—RX-Type, (2-12), (1)

TYPE=J
coded the same as the standard form.

MODE=24|31
coded the same as the standard form. This specification must match that of the list form.

PDAB—Construct a parallel data access block (QSAM)

The PDAB macro is used with the GET (TYPE=P) macro. It defines an area in the problem program where the open and close routines build and maintain a queue of DCB addresses for use by the get routine.

The parallel data access block is constructed during the assembly of the problem program. MAXDCB must be coded in the PDAB macro, because it cannot be supplied from any other source.

Certain data set characteristics prevent a DCB address from being available on the queue—see the description of QSAM parallel input processing in *z/OS DFSMS Using Data Sets*.

Restriction: Do not use the PDAB macro if a QSAM GET will be used in 31-bit addressing mode.

UNIX files: OPEN ignores the PDAB for a DCB that is for a UNIX file or subsystem data set.

The format of the PDAB macro is:

[*label*]	PDAB	MAXDCB=*absexp*

MAXDCB=*absexp* (maximum value is 32767 bytes)
specifies the maximum number of DCBs that you require in the queue for a GET request.

The number of bytes required for PDAB is equal to 24+8n, where n is the value of the keyword, MAXDCB.

PDABD—Provide symbolic reference to a parallel data access block (QSAM)

The PDABD macro generates a dummy control section that provides symbolic names for the fields in one or more parallel data access blocks. The names, attributes, and descriptions of the fields appear in "PDABD symbolic field names" on page 294.

The name of the dummy control section generated by a PDABD macro is IHAPDAB.A USING instruction specifying IHAPDAB and a dummy section base register containing the address of the actual parallel data access block should come before any of the symbolic names provided by the dummy section.You may code the PDABD macro once in any assembled module. However, you can use the resulting symbolic names for any number of parallel data access blocks by changing the address in the dummy section base register. You can code the PDABD macro at any point in a control section. If coded at any point other than at the end of a control section, the control section must be resumed by coding a CSECT instruction.

The format of the PDABD macro is b PDABD b

PDABD symbolic field names

The following describes PDABD fields of the dummy control section generated by the PDABD macro. Included are the names, attributes, and descriptions of the dummy control section.

```
          PDABD
IHAPDAB   DSECT
PDANODCB  DS       H   Number of DCB addresses in list.
PDAMAXCB  DS       H   Maximum number of addresses allowed.
          DS       A   Reserved for IBM use.
          DS       F   Reserved for IBM use.
PDADCBLA  DS       A   Address of the last DCB entry.
PDADCBEP  DS       A   Address of DCB entry last processed.
          DS       F   Reserved for IBM use.
PDADCBAL  EQU      *   Start of DCB list.
```

POINT—Position for access (BPAM and BSAM—tape and DASD only)

The POINT macro causes the next READ or WRITE operation to be for the specified data set block on the current volume for BSAM or on the current data set for BPAM. With BPAM concatenation, you can switch to a different data set with the FIND macro. Before you issue the POINT macro, test for completion of all input and output operations using the same data control block.If you are processing a data set opened for UPDAT, the next operation against the DCB after the POINT macro must be a READ macro. If you are processing an output data set, the next operation against the DCB after the POINT macro must be

a WRITE macro before you close the data set, unless you have already issued the CLOSE macro (with TYPE=T specified) before the POINT macro.

The POINT macro with the REL parameter, which is the default, works with any magnetic tape drive. However, POINT with the ABS parameter works only on cartridge tapes, such as the 3480, 3490 and 3590.

The POINT macro can be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

When a PDSE is open for output, if you use the POINT macro to position to a member other than the member currently processing, it results in an I/O error on the next write.

If you issued a CLOSE TYPE=T and are not open for INPUT, UPDAT, or RDBACK and are positioned to other than the end of the data set but do not want to truncate it, you must reposition to the end of the data set before closing it.

POINT positions to the first segment of a spanned record even if the NOTE was done on another segment. If the current record spans blocks, set the z byte of the TTR field to one to access the next record (not segment).

If you set BLOCKTOKENSIZE=LARGE (DCBELARGE=1) and MACRF and DSORG in the DCB indicate BSAM and the NOTE and POINT macros, then the OPEN macro allows the opening of large format data sets. BLOCKTOKENSIZE=LARGE signifies that you aware of the changes in the DSCB, DEB, and TTR conversion routines and that the NOTE and POINT macros must use the TTTR interface in place of the TTR0 interface regardless of what kind of DASD the data set resides on.

When you code BLOCKTOKENSIZE=LARGE on the DCBE with BSAM or BPAM then:

- For a sequential non-extended format data set or PDS, NOTE and POINT use TTTR rather than TTRx. In a PDS, the high order byte is always 0. There is no effect on the FIND macro which continues to accept a four byte TTRc where c is the concatenation number as received from BLDL. The content of DCBRELAD and DCBERELA, which identify the location of the beginning of the member, remains unchanged.
- For extended format, compressed format, or UNIX files, the relative block number is expressed in four bytes rather than the three high order bytes. This removes a size restriction for the NOTE and POINT macros.
- You receive a four byte simulated TTTR value instead of the previous three byte field. The maximum number of records increase from 15,728,639 to 4,277,145,599.

  Note: If you pass to POINT a TTR returned from BLDL you must right align the three bytes returned into the four byte field that you pass.

Using Spooled Data Sets: The (NOTE and) POINT macros cannot be used with spooled data sets.

Using Subsystem Data Sets: A subsystem data set is represented by a DD statement that has the SUBSYS keyword.

The NOTE and POINT macros with TYPE=REL specified or defaulted can be used for subsystem data sets if the subsystem supports it. Assume it does not work unless the subsystem documentation says it is supported. If the subsystem does not support it, the results are unpredictable.

Using UNIX Files: The POINT macro can be issued for UNIX files, except for FIFO or character special files or with PATHOPTS=OAPPEND.

Using POINT with PDSEs: The POINT macro establishes a connection to the PDSE member and the connection is maintained until the PDSE is closed. The POINT macro can start the next READ or WRITE operation at the beginning of a member or anywhere within a member. To position to a record within another member, issue a POINT or FIND macro to the beginning of that member, then issue another POINT to position to the record you want. You cannot position from one PDSE member to a record other than the first block in another member because either data from the first member record is read, or an I/O error occurs.

The standard form of the POINT macro is:

[*label*]	POINT	*dcb address* ,*block address* [,TYPE={ABS|REL|RELNEXT}]

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for the opened data set to be positioned.

***block address*—RX-Type Address, (2-12), or (0)**
indicates which block in the data set is processed next.

If TYPE=ABS is Specified: For a magnetic tape drive, when TYPE=ABS is specified, the *block address* specifies the address of a fullword on a fullword boundary that contains the physical block identifier of the block in the data set that is to be processed next. If you code (0), it means register zero contains the block identifier and not the address. Do not code a reference to register 0 with a symbol; it gives unpredictable results. This physical block identifier is provided as output from a prior execution of the NOTE macro.

If TYPE=REL is Specified: When TYPE=REL is specified or defaults, the *block address* specifies the address of a fullword on a fullword boundary that contains the relative address of the block in the data set that is to be processed next.

The first block of a magnetic tape data set is always specified by the hexadecimal value 0000 0001. You can specify the first block of a direct access storage device data set with hexadecimal 0000 0001 (except for PDSEs or a UNIX member with BPAM). If you do not code BLOCKTOKENSIZE=LARGE on your DCBE macro, then you can specify the first block of a direct access storage device data set with 0000 0100 (except for a PDSE or a UNIX member with BPAM).

Specify the relative address as follows.

Magnetic Tape: The block number is in binary and right-adjusted in the fullword with the high-order bits set to 0; add 1 if reading tape backward.

Do not use the POINT macro for tapes without standard labels when either one of these situations exist:

- The data set is opened for RDBACK, EXTEND or OUTINX
- The data set is opened for OUTPUT or OUTIN and the DD statement for the data set specifies DISP=MOD

If OPTCD=H is indicated in the data control block, you can use the POINT macro to perform record positioning on Virtual Storage Extended (VSE) tapes (formerly called DOS tapes) that contain embedded checkpoint records. Any embedded checkpoint records found during the record positioning are bypassed and not counted as blocks spaced over. OPTCD=H must be specified in a job control language DD statement. Do not use the POINT macro to backspace VSE 7-track tapes that are written in data convert mode and that contain embedded checkpoint records.

When an end-of-data condition is reached on magnetic tape, you must first reposition the tape for processing your data set. Then, you can issue the POINT macro; otherwise, the POINT operation fails . (Issuing CLOSE TYPE=T is an easy method to use to accomplish repositioning in your EODAD routine.)

The first block of a magnetic tape data set is always specified by the hexadecimal value 0000 0001.

Direct Access Storage Devices: The second parameter that your program supplies to the POINT macro is called a block token or block address. You can always supply to POINT a token that your program received from NOTE for the same DCB. The token's format depends on the type of data set and on whether you coded BLOCKTOKENSIZE=LARGE on the DCBE macro that the DCB points to. If your program turns on the DCBELARGE bit in the DCBE, it has the same effect as coding BLOCKTOKENSIZE=LARGE.

If you do not code BLOCKTOKENSIZE=LARGE, then the word that your program passes contains the block token in the three high order bytes and the low order byte contains X'00'. If you code BLOCKTOKENSIZE=LARGE, then your program passes a block token of four bytes.

BLOCKTOKENSIZE=LARGE has no effect on the FIND and BLDL macros or on the content of DCBRELAD and DCBERELA, which identify the location of the beginning of the member. They always use a three-byte member token.

If your program runs on a system before z/OS 1.7, then the system ignores any setting of BLOCKTOKENSIZE=LARGE.

Positioning to the Next Block:

You can position to a block by supplying the token for the previous block. If you do not code BLOCKTOKENSIZE=LARGE, then you set the low order byte of the word to X'01' to position to the next block or you can use the TYPE=RELNEXT option. If you code BLOCKTOKENSIZE=LARGE, then code the TYPE=RELNEXT parameter on the POINT macro to position to the next block.

Using POINT with a basic format data set, large format data set or PDS. Use either of these two forms:

TTRz
Use this form if you do not code BLOCKTOKENSIZE=LARGE.

The two TT bytes contain the relative track number in the data set, where the first track is X'0000'.
The R byte is the relative block number on that track, where the first block is X'01'. If the z byte is X'00', position to the specified block. If the z byte is X'01', position to the following block. The first block of a data set is either X'00000100' or X'00000001'.

TTTR
Use this form if you code BLOCKTOKENSIZE=LARGE. The three TTT bytes contain the relative track number in the data set, where the first track is X'000000', The R byte is the relative block number on that track, where the first block is X'01'. To position to the following block, use TYPE=RELNEXT.

If the data set is large format, your program must use this form unless less than 65536 tracks are allocated to it on the volume and your program opened the data set with the INPUT option. If the data set is a PDS or a basic format data set, then the high order byte always contains X'00' because those data sets cannot contain more than X'FFFF' tracks. If you pass to POINT a TTR returned from BLDL, you must right align the three byte field that you pass.

Using POINT with a PDSE or UNIX file with BPAM: The three or four byte token does not represent the physical location of the data set or member. Without BLOCKTOKENSIZE=LARGE, the maximum number of records in a member is 15,728,639. With BLOCKTOKENSIZE=LARGE, the maximum number of records in a member is 4,277,145,599. You cannot use POINT with a UNIX FIFO or character special file.

Unless preceded by another POINT, a POINT to an invalid value in a UNIX file gives an I/O error for the next READ or WRITE.

In a binary UNIX file with RECFM=V(B) or RECFM=U, a POINT to other than the first block results in an abend.

Using POINT with a UNIX file with BSAM: You can use the POINT macro for UNIX files, except for FIFO or character special files or with PATHOPTS=OAPPEND. The block token is the relative record number (RRN) from the beginning of the file. If you do not code BLOCKTOKENSIZE=LARGE, then (1) POINT does not support a UNIX file that contains more than 16 mega-records minus two (X'000000'-2 or 16,777,214) and (2) a POINT macro to after 16 mega-records minus two returns an invalid value (X'FFFFFF'). If you code BLOCKTOKENSIZE=LARGE, then you do not have this limit.

A POINT to a location past the end of the file or after a block that follows the most recent WRITE gives an I/O error for the next READ or WRITE.

Unless preceded by another POINT, a POINT to an invalid value gives an I/O error for the next READ or WRITE.

Using POINT with an extended format data set or compressed format data set: The three or four-byte value is the relative block number within the data set. If you do not code

BLOCKTOKENSIZE=LARGE on the DCBE macro, then your program can address up to 16,777,215 blocks. If you code BLOCKTOKENSIZE=LARGE, then your program can address up to 4,294,967,295 blocks.

In a compressed format data set, the system writes your data in blocks whose sizes are not under your control but the system maintains logical block sizes as seen by your program. Therefore, the relative block number identifies the uncompressed block.

Your program can derive the input to POINT from the value returned by the NOTE macro. Your program can modify that value by setting the low order byte (the Z byte) to 1 in order to obtain the block following the block defined by that NOTE value.

If you issue a POINT to a location past the end of the data set or after the block that follows the most recent WRITE, the next READ or WRITE results in an I/O error. A POINT issued immediately following an open for output (while positioned to the beginning of the data set) causes the next WRITE to result in an I/O error. Also, whenever an I/O error is encountered, any further POINTs causes the subsequent READ or WRITES to result in an I/O error.

Using POINT with UNIX files: POINT is supported for UNIX files, except for FIFO or character special files, or when PATHOPTS=OAPPEND.

The TTRz passed to POINT should be a token derived from NOTE. You can modify the token by setting the low order byte (the z byte) to 1 to get the block following the block defined by the token. The token is the relative record number (RRN) from the beginning of the file.

A POINT to a location past the end of the file or after a block that follows the most recent WRITE gives an I/O error for the next READ or WRITE.

Unless preceded by another POINT, a POINT to an invalid value gives an I/O error for the next READ or WRITE.

In a binary file with RECFM=V(B) or RECFM=U,a POINT to other than the first block results in an abend.

TYPE={ABS|REL|RELNEXT}
indicates whether the *block address* is a physical block identifier or a relative address.

ABS
indicates that the *block address* specifies an address of a fullword on a fullword boundary containing a physical block identifier of the block in the data set that is to be processed next. This option is only for a cartridge tape.

REL
indicates that the *block address* specifies an address of a fullword on a fullword boundary containing the relative address of the block in the data set that is to be processed next. This option is for DASD (including UNIX) or tape.

RELNEXT
indicates that the *block address* specifies the address of a word containing the block token of the block that precedes the block that is to be processed next. If your DCBE does not have BLOCKTOKENSIZE=LARGE, then RELNEXT has the same effect as setting the low order byte of the block token to X'01'.

If the volume cannot be positioned correctly or if the block identification is not of the correct format, the error analysis (SYNAD) routine is given control when the next CHECK macro is executed.

POINT completion codes

When the system returns control to your problem program and you have specified the ABS parameter, the low-order byte of register 15 contains a return code. If return code = 08, the low-order byte of register 0 contains a reason code.

The POINT return and reason codes are:

If TYPE=ABS is specified

Return Code (15)	Reason Code (0)	Meaning
00 (X'00')		Successful completion.
04 (X'04')		Device does not support block identifier.
08 (X'08')	01 (X'01')	Incorrect parameter.
08 (X'08')	02 (X'02')	Incorrect DCB or a DEB error.
08 (X'08')	03 (X'03')	Environmental error.
08 (X'08')	11 (X'0B')	Unsuccessful call to ESTAE macro.
08 (X'08')	12 (X'0C')	Insufficient virtual storage available.
12 (X'0C')		Input/output error.

If TYPE=REL is specified

None.

POINT TYPE=ABS—List form

You can use the list form of the POINT macro when you code TYPE=ABS. This list form constructs a parameter list.

The description of the standard form of the POINT macro explains the function of each parameter. The format description below shows the optional and required parameters in the list form only.

The list form of the POINT macro is:

[*label*]	POINT	[,*block number*] ,TYPE=ABS ,MF=L

block number—absolute arithmetic expression. It is the absolute block identifier, not its address. You can code a symbolic expression. It can contain a hexadecimal value.

TYPE=ABS
indicates that the block number is a physical block identifier.

MF=L
specifies that the POINT macro is used to create a parameter list for the POINT macro with TYPE=ABS.

POINT TYPE=ABS—Execute form

You can use the execute form of the POINT macro when you code TYPE=ABS. The execute form uses and can modify a parameter list that is generated by the list form.

The description of the standard form of the POINT macro explains the function of each parameter. The format description below shows the optional and required parameters in the execute form only.

The execute form of the POINT macro is:

[*label*]	POINT	*dcb address* [,*block number*] ,TYPE=ABS ,MF=(E,*list-address*)

dcb address—RX-type address, (2-12)

block address—RX-type address, (2-12) or (0).

TYPE=ABS
indicates that the block number is a physical block identifier.

MF=(E,list-address)
specifies that the execute form of the POINT macro, and that an existing parameter list that was generated with MF=L. Initialize the parameter list before executing the execute form. Specify block address in either or both forms.

PRTOV—Test for printer carriage overflow (BSAM and QSAM—online printer and 3525 card punch)

The PRTOV macro controls the page format for a directly-allocated printer when carriage control characters are not used or to supplement the carriage control characters being used. A directly-allocated (online) printer is allocated to the application program and is not a spooled data set.

The PRTOV macro tests for an overflow condition on the specified channel (either channel 9 or channel 12) of the printer carriage control, and either skips the printer carriage to the line corresponding to channel 1, or transfers control to the exit address, if one is specified. Overflow is detected after printing the line that follows the line corresponding to channel 9 or channel 12. You should issue the PRTOV macro each time you want the system to test for an overflow condition.

When the PRTOV macro is used with a 3525 card punch, print feature, channel 9 or 12 can be tested. If an overflow condition occurs, control is passed to the overflow exit routine if the overflow exit address is coded, or a skip to channel 1 (first print-line of the next card) occurs.

When requesting overprinting (for example, to underscore a line), issue the PRTOV macro before the first PUT or WRITE macro only. The PRTOV macro is useful only for directly-allocated printers. PRTOV has no effect for other devices, such as SYSOUT data sets or the 3525 card punch without the printing feature. You cannot use PRTOV to request overprinting on the 3525. The effect of overprinting differs for various printer models. See the appropriate device reference manual.

The PRTOV macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The format of the PRTOV macro is:

[*label*]	PRTOV	*dcb address* ,{9|12} [,*overflow exit address*]

***dcb address*—RX-Type Address or (2-12)**
specifies the address of the data control block opened for output to a directly-allocated printer or 3525 card punch with a print feature.

9
12
These parameters specify the channel to be tested by the PRTOV macro. For a directly-allocated printer, 9 and 12 correspond to carriage control channels 9 and 12. For the 3525 card punch, 9 corresponds to print line number 17, and 12 corresponds to print line number 23.

***overflow exit address*—RX-type address or (2-12)**
specifies the address of the user-supplied routine given control when an overflow condition is detected on the specified channel. If this parameter is omitted, the printer carriage skips to the first line of the next page or the 3525 skips to the first line of the next card before executing the next PUT or WRITE macro.

The overflow exit routine receives control in the addressing mode in which you issue the PRTOV macro. If you issue PRTOV in 31-bit addressing mode, the overflow exit routine may reside above the 16MB line.

When the overflow exit routine is given control, the contents of the registers are as follows:

Register	Contents
0 and 1	The contents are destroyed.
2 - 13	The same contents as before the macro was executed.
14	Return address.
15	Overflow exit routine address.

PUT—Write next record (QISAM interface to VSAM)

The PUT macro writes a record into an indexed sequential data set. If the move mode is used, the PUT macro moves a logical record into an output buffer from which it is written. If locate mode is specified, the address of the next available output buffer segment is available in register 1 after the PUT macro is executed. The logical record can then be constructed in the buffer for output as the next record.

The records are blocked by the system (if specified in the data control block) before being placed in the data set. The system uses the length specified in the record length (DCBLRECL) field of the data control block as the length of the record currently being written.When constructing blocked variable-length records in the locate mode, the problem program might either specify the maximum record length once in the DCBLRECL field of the data control block or provide the actual record length in the DCBLRECL field before issuing each PUT macro. Using the maximum record length may result in more but shorter blocks, because the system uses this length when it tests to see if the next record can be contained in the current block.

The PUT macro is used to write a new indexed sequential data set or extend it.To extend the data set, the key of any added record must be higher than the highest key existing in the data set, and the disposition parameter of the DD statement must be specified as DISP=MOD. The new records are placed in the prime data space, starting in the first available space, until the original space allocation is exhausted.

To allocate a data set using previously allocated space, the disposition parameter of the DD statement must specify DISP=OLD.

For QISAM, interface to VSAM PUT must be issued in 24-bit mode.

Recommendation: The system no longer supports indexed sequential data sets. Convert the data set to a key sequenced data set (KSDS) and use the ISAM interface of VSAM or convert your program to use VSAM.

The format of the PUT macro is:

[*label*]	PUT	*dcb address* [,*area address*]

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for the opened indexed sequential data set.

***area address*—RX-Type Address, (2-12), or (0)**
specifies the address of the area containing the record to be written (move mode only). Either move or locate mode can be used with QISAM interface to VSAM, but they must not be mixed in the specified data control block. The following describes operations for locate and move modes:

Locate Mode: If locate mode is specified in the data control block, the *area address* must be omitted.The system returns the address of the next available buffer in register 1. This is the buffer into which you should move the next record. The record is not written until another PUT macro is issued for the same data control block or a CLOSE macro is issued to close the data set.

Move Mode: If move mode has been specified in the data control block, the *area address* must specify the address in the problem program that contains the record to be written.The system moves the record from the area to an output buffer before control is returned. If the *area address* is omitted, the system assumes that register 0 contains the area address.

PUT routine exit

The error analysis (SYNAD) routine is given control if the output operation cannot be completed satisfactorily. The contents of the registers when the error analysis routine is given control are described in "Status information following an input/output operation" on page 373.

PUT—Write next record (QSAM)

The PUT macro writes a record in a sequential data set, partitioned data set, PDSE or UNIX file. You can specify locate mode, move mode, and data mode in the DCB macro. In the locate mode, the PUT macro returns the address of an area in an output buffer in register 1. You should then construct, at this address, the next sequential record or record segment. In the move mode, the PUT macro moves a logical record into an output buffer. In the data mode, which is available only for variable-length spanned records, the PUT macro moves only the data portion of the record into one or more output buffers.

The access method blocks records as specified in the data control block before being placed in the data set. *Blocking* means collecting records into blocks.

This is how the system determines the length of the record:

- For variable-length records when using locate mode, PUT uses the DCBLRECL field to locate a buffer segment of sufficient size, but the next PUT verifies the length of the record actually constructed before the record is written. You can fill the output block to the maximum if, before issuing the PUT macro, you set DCBLRECL equal to the length of the next record).
- For variable-length records when using move mode, PUT uses the RDW to verify that the record will fit in a QSAM buffer.
- For variable length spanned records, the system segments the record according to the record length, buffer length, and amount of unused space remaining in the output buffer. The smallest segment created is 5 bytes, 4 for the segment descriptor word plus 1 byte of data.
- For undefined-length records without LBI, you set the DCBLRECL field before each PUT to determine the length of the next record. Use the DCBD macro to map DCBLRECL.
- For undefined-length records with LBI, you set DCBEBLKSI before each PUT to determine the length of the next record. Use the IHADCBE macro to map DCBEBLKSI.
- For fixed-length records, all records are the length specified when the DCB was opened.

The PUT macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses. This includes allowing the caller to issue QSAM macros in 31-bit addressing mode regardless of whether the buffers are above or below the 16 MB line. Most types of data sets support 31-bit mode. See "Environmental considerations" on page xxii.

QSAM allows data areas to be located above the 16MB line. To take advantage of providing data areas above the 16MB line for QSAM macros, the issuer of the PUT macro must then execute in 31-bit addressing mode. To take advantage of QSAM buffers above the line, you must specify for OPEN to obtain the buffers above the line and the issuer of the PUT macro must then execute in 31-bit addressing mode. To specify that OPEN is to get buffers above the 16MB line, code RMODE31=BUFF on the DCBE macro.

Data Conversion: You can request conversion by coding LABEL=(,AL) or (,AUL) in the DD statement, or by coding OPTCD=Q in the DCB macro or DCB subparameter of the DD statement.When conversion is requested, all QSAM records whose record format (RECFM parameter) is F, FB, D, DS, DB, DBS, or U are automatically converted from one character representation to another. Conversion is performed according to one of the following techniques:

- **Coded Character Set Identifier (CCSID) Conversion**

If CCSIDs are supplied from any source for ISO/ANSI V4 tapes, records are converted from the CCSID as seen by the problem program to the CCSID which represents the data on tape. You can also prevent conversion by supplying a special CCSID. CCSID may be supplied in the CCSID subparameter of a JOB, EXEC, or DD statement or the tape label.

- **Default Character Conversion**

  If you are using non-ISO/ANSI V4 tapes or if CCSIDs are not supplied by any source, data management converts the records from EBCDIC code to ASCII code using specific tables defined for this default character conversion.

Refer to *z/OS DFSMS Using Data Sets*, for a complete description of CCSID conversion and default character conversion.

The format of the PUT macro is:

[*label*]	PUT	*dcb address* [,*area address*]

***dcb address*—RX-type address, (2-12), or (1)**
specifies the address of the data control block for the data set opened for output.

***area address*—RX-type address, (2-12), or (0)**
specifies the address of an area containing the record to be written (move or data mode). The move, locate, or data mode can be used with QSAM, but they must not be mixed in the specified data control block. When issued in 31-bit addressing mode, the input area address (move or data mode) must be clean 31-bit addresses. For move or data mode, if the input area address resides above the 16MB line, you must issue the PUT in 31-bit mode. If you requested that OPEN get buffers above the 16MB line, the PUT must be issued in 31-bit mode. If the *area address* is omitted in the move or data mode, the system assumes that register zero contains the area address. The following describes the operation of the three modes:

Locate Mode: If you specify locate mode, omit the *area address*. The system returns the address of the next available buffer in register 1. This is the buffer into which your program later places the next record.

When variable-length spanned records are processed without the extended logical record interface (XLRI), and a record area is provided for a logical record interface (LRI) (BFTEK=A has been specified in the data control block or a BUILDRCD macro has been issued), the address returned in register 1 points to an area large enough to contain the maximum record size (up to 32756 bytes). The system segments the record and writes all segments, providing proper control codes for each segment. If, for variable-length spanned records, a record area has not been provided, the actual length remaining in the buffer is returned in register 0. In this case, you must segment the records and process them in record segments. ISO/ANSI spanned records, RECFM=DS or RECFM=DBS, cannot be processed in segment mode. The record or segment is not written until another PUT macro is issued for the same data control block or an FEOV or CLOSE macro is issued.

When a PUT macro is used in the locate mode, the address of the buffer for the first record or segment is obtained by issuing a PUT macro after open. QSAM returns the address in register 1. Then, move data to this address. The buffer is not written to the data set until the next PUT macro is issued. If records are blocked, the data is not written to the data set until the PUT following the one that filled the buffer. Each PUT macro returns the address of the next buffer in register 1. After this address is given to you, QSAM always counts this address as a valid record. You should always place valid data at the address returned in register 1 before issuing another PUT or FEOV or CLOSE macro. Otherwise, residual data at that location is written to the data set. After issuing an FEOV macro (for multivolume data sets), you must reinitialize register 1 with the first buffer address for the next volume by issuing a PUT macro after return from FEOV.

Move Mode: If move mode is specified in the data control block, the *area address* specifies the address of the area containing the record to be written.The system moves the record to an output buffer before control is returned.

Data Mode: If data mode is specified in the data control block (data mode can be specified for variable-length spanned records only), the *area address* specifies the address of an area in the problem program that contains the data portion of the record to be written. The system moves the data portion of the record to an output buffer before control is returned. You must place the total data length in the DCBPRECL (not the DCBLRECL) field of the data control block before issuing the PUT macro. On return from the PUT, the value stored in DCBPRECL may have been modified by the system.

Extended Logical Record Interface (XLRI): When the PUT macro is used with the extended logical record interface, the address returned in register 1 points to an area used to build a 4-byte logical record length field (RDW) followed by a complete logical record. The logical record length byte count occupies the 3 low-order bytes of the record length field and must include the length of the field. The high-order byte must be zero. The DCB LRECL value indicates the length of the longest logical record of the data set in 'K' (1024-byte) units.

PUT routine exit

If the output operation cannot be completed satisfactorily due to an uncorrectable I/O error, the error analysis (SYNAD) routine is given control after a later PUT instruction is issued. The contents of the registers when the error analysis routine is given control are described in *z/OS DFSMS Using Data Sets*.

If your SYNAD routine is entered, it is entered in the addressing mode in which the PUT was issued. If you supplied a SYNAD routine which resides above the line in the DCBE, then the PUT must be issued in 31-bit addressing mode. On entry to the SYNAD routine, register 1 contains error flags in byte 0 followed by the DCB address in bytes 1-3. For 31-bit callers, the caller must save the error flags, if needed, and then clear the high order byte of register 1 before using it to access fields within the DCB in the SYNAD routine.

PUTX—Write a record from an existing data set (QISAM interface to VSAM and QSAM)

The PUTX macro returns an updated record to a data set (QISAM interface to VSAM and QSAM) or writes a record from an input data set into an output data set (QSAM only). There are two modes of the PUTX macro. The output mode (QSAM only) allows writing a record from an input data set on a different output data set. The output data set can specify the spanning of variable-length records, but the input data set must not contain spanned records.

The update mode returns an updated record to the data set from which it was read. The logical records are blocked by the control program, as specified in the data control block, before they are placed in the output data set. The control program uses the length specified in the DCBLRECL field as the length of the record currently being stored. Control is not returned to your user program until the control program processes the record.

For SYSOUT data sets, the PUTX macro can be used only in the output mode.

The record descriptor word in variable-length records must not be changed.

The PUTX macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses. This includes allowing the caller to issue QSAM macros in 31-bit addressing mode regardless of whether the buffers are above or below the 16MB line. Most types of data sets support 31-bit mode. See “Environmental considerations” on page xxii.

QSAM allows data areas to be located above the 16MB line. To take advantage of providing data areas above the 16MB line for QSAM macros, the issuer of the PUTX macro must then execute in 31-bit addressing mode. To take advantage of QSAM buffers above the line, you must specify for OPEN to obtain the buffers above the line and the issuer of the PUTX macro must then execute in 31-bit addressing mode. To specify that OPEN is to get buffers above the 16MB line, code RMODE31=BUFF on the DCBE macro.

The format of the PUTX macro is:

[*label*]	PUTX	*dcb address* [, *input dcb address*]

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for a data set opened for output.

***input dcb address*—RX-Type Address, (2-12), or (0)**
specifies the address of a data control block opened for input. When issued in 31-bit addressing mode, the input DCB address must be a clean 31-bit address. If you requested that OPEN get buffers above the 16MB line, the PUTX must be issued in 31-bit mode. The PUTX macro can be used for the following modes:

Output Mode: This mode is used with QSAM only. The *input dcb address* specifies the address of the data control block opened for input. If this parameter is omitted, the system assumes that register 0 contains the input dcb address.

Update Mode: The *input dcb address* is omitted for update mode.

PUTX routine exit

The error analysis (SYNAD) routine is given control if the operation is not completed satisfactorily due to an uncorrectable I/O error. The contents of the registers when the error analysis routine is given control are described in *z/OS DFSMS Using Data Sets*.

If your SYNAD routine is entered, it is entered in the addressing mode in which the PUTX was issued. If you supplied a SYNAD routine which resides above the line in the DCBE, the PUTX must be issued in 31-bit addressing mode. On entry to the SYNAD routine, register 1 contains error flags in byte 0 followed by the DCB address in bytes 1-3. For 31-bit callers, the caller must save the error flags, if needed, and then clear the high order byte of register 1 before using it to access fields within the DCB in the SYNAD routine.

READ—Read a block (BDAM)

The READ macro retrieves a block from a data set and places it in a designated area of storage. Control might be returned to the problem program before the block is retrieved.The input operation must be tested for completion using a CHECK or WAIT macro. A data event control block, shown in "Status information following an input/output operation" on page 373, is constructed as part of the macro expansion.

Addressing mode: The READ macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The standard form of the READ macro is written as follows (the list and execute forms are shown following the descriptions of the standard form):

[*label*]	READ	*decb name* ,{DI[F|X][R|RU]} {DK[F|X][R|RU]} ,*dcb address* ,{*area address*|'S'} ,{*length*|'S'} ,{*key address*|'S'|0} ,*block address* [,*next address*]

***decb name*—symbol**
specifies the name that is assigned to the data event control block that is created as part of the macro expansion.

***type*—{DI[F|X][R|RU]}**
{DK[F|X][R|RU]}
is coded in one of the combinations shown above to specify the type of read operation and the optional services performed by the system:

DI
specifies that the data and key, if any, are read from a specific device address. The device address, which can be designated by any of the three addressing methods, is supplied by the *block address*.

DK
specifies that the data (only) is read from a device address identified by a specific key. The key used as a search argument must be supplied in the area specified by the *key address*. The search for the key starts at the device address supplied in the area specified by the *block address*. The description of the DCB macro, LIMCT, contains a description of the search.

F
requests that the system provide block position feedback into the area specified by the *block address*. This character can be coded as a suffix to DI or DK as shown above.

X
requests exclusive control of the data block being read, and that the system provide block position feedback into the area specified by the *block address*. If OPTCD=F is not specified, the feedback is provided in the form of an 8-byte absolute address (MBBCCHHR). The descriptions of the WRITE and RELEX macros contain a description of releasing a data block under exclusive control. This character can be coded as a suffix to DI or DK as shown above.

R
requests that the the system provide next address feedback into the area specified by *next address*. When R is coded, the feedback is the relative track address of the next data record. This character can be coded as a suffix to DI, DK, DIF, DIX, DKF, or DKX as shown above, but can be coded only for use with variable-length spanned records.

RU
requests that the the system provide *next address* feedback into the area specified by the *next address*. When RU is coded, the feedback is the relative track address of the next capacity record (R0) or data record whichever occurs first. These characters can be coded as a suffix to DI, DK, DIF, DIX, DKF, or DKX, but it can be coded only for use with variable-length spanned records.

***dcb address*—A-Type Address or (2-12)**
specifies the address of the data control block opened for the data set to be read.

***area address*—A-Type Address, (2-12), or 'S'**
specifies the address of the area in which the data block is to be placed. If `'S'` is coded instead of an address, dynamic buffering is requested (dynamic buffering must also be specified in the MACRF parameter of the DCB macro). When dynamic buffering is used, the system acquires a buffer and places its address in the data event control block.

***length*—symbol, decimal digit, absexp, (2-12), or 'S'**
specifies the number of data bytes to be read up to a maximum of 32760. If `'S'` is coded instead of a length, the number of bytes to be read is taken from the data control block. If neither *length* nor `'S'` is specified, no error indication is given when your program is assembled, but your program must insert a length into the data event control block (DECB) before the READ is issued.

***key address*—A-Type Address, (2-12), 'S', or 0**
specifies the address of the area for the key of the desired data block. If the search operation is made using a key, the area must contain the key. Otherwise, the key is read into the designated area. If the key is read and `'S'` was coded for the *area address*, you can also code `'S'` for the key address; the key and data are read sequentially into the buffer acquired by the system. If the key is not to be read, specify `0` instead of an address or `'S'`.

***block address*—A-Type Address or (2-12)**
specifies the address of the area containing the relative block address, relative track address, or actual device address of the data block to be retrieved. The device address of the data block retrieved is placed in this area if block position feedback is requested. The length of the area containing the address depends on whether the feedback option (OPTCD=F) is specified in the data control block and if the READ macro requested feedback.

If OPTCD=F is specified, feedback (if requested) is in the same form as originally presented by the READ macro, and the field can be either 3 or 8 bytes long, depending on the type of addressing.

If OPTCD=F is not specified, feedback (if requested) is as an actual device address, and the field must be 8 bytes long.

***next address*—A-Type Address or (2-12)**
specifies the address of the storage area in which the system places the relative address of the next block. *Length* must be specified as 'S'. When *next address* is specified, an R or RU must be added to the *type* parameter (for example, DIR or DIRU). The R indicates that the next address returned is the next data record. RU indicates that the next address returned is for the next data or capacity record, whichever occurs first. The *next address* parameter can be coded only for use with variable-length spanned records.

READ—Read a block of records (BISAM interface to VSAM)

The READ macro retrieves an unblocked record, or a block containing a specified logical record, from a data set. The block is placed in a designated area of storage, and the address of the logical record is placed in the data event control block. The data event control block is constructed as part of the macro expansion and is described in "Status information following an input/output operation" on page 373.

Control might be returned to the problem program before the block is retrieved. The input operation must be tested for completion using a WAIT or CHECK macro.

The READ macro for BISAM interface to VSAM must be issued in 24-bit mode.

Recommendation: The system no longer supports indexed sequential data sets. Convert the data set to a key sequenced data set (KSDS) and use the ISAM interface of VSAM or convert your program to use VSAM.

The standard form of the READ macro is written as follows for BISAM (the list and execute forms are shown following the descriptions of the standard form):

[*label*]	READ	*decb name* ,{K|KU} ,*dcb address* ,{*area address*|'S'} ,{*length*|'S'} ,*key address*

***decb name*—symbol**
specifies the name that is assigned to the data event control block (DECB) that is created as part of the macro expansion.

***type*—{K|KU}**
is coded as shown to specify the type of read operation:

K
specifies normal retrieval.

KU
specifies that the record retrieved is updated and returned to the data set. The system saves the device address to be returned.

When an indexed sequential data set is being updated with a READ KU macro and a WRITE K macro, both the READ and WRITE macros must refer to the same data event control block. This update operation can be performed by using a list-form instruction to create the list (data event control block) and by using the execute form of the READ and WRITE macros to refer to the same list.

***dcb address*—A-Type Address or (2-12)**
specifies the address of the data control block for the opened data set to be read.

***area address*—A-Type Address, (2-12), or 'S'**
specifies the address of the area in which the data block is placed. The first 16 bytes of this area are used by the system and do not contain information from the data block. The *area address*

must specify a different area than the key address. Dynamic buffering is specified by coding 'S' instead of anaddress. The address of the acquired storage area is returned in the data event control block.Indexed sequential buffer and work area requirements are described in *z/OS DFSMS Using Data Sets*.

***length*—symbol, decimal digit, absexp, (2-12), or 'S'**
specifies the number of bytes to be read up to a maximum of 32760. If 'S' is coded instead of a length, the number of bytes to be read is taken from the count field of the record. For blocked records, 'S' *must* be coded.

***key address*—A-Type Address or (2-12)**
specifies the address of the area in the problem program containing the key of a logical record in the block to be retrieved. When the input operation is complete, the storage address of the logical record is placed in the data event control block. The *key address* must specify a different area than the *area address.*

READ—Read a block (BPAM and BSAM)

The READ macro retrieves a block from a data set and places it in a designated area of storage (input area). Control might be returned to the problem program before the block is retrieved. The input operation must be tested for completion using a CHECK, WAIT or EVENTS macro. After completion, you can determine the length of the block by using a technique described in *z/OS DFSMS Using Data Sets*.

If you are using the large block interface (OPEN set DCBESLBI on) for fixed-length blocked or undefined-length records, the actual length of the record block that was read can be found in a 4-byte length-read field. Locate the length-read field by performing the following tasks:

1. After issuing the CHECK, WAIT or EVENTS macro for the DECB for the READ request, obtain the status area address from the word that is 16 bytes from the start of the DECB.
2. Subtract 12 from the address of the status area to obtain the address of the length-read field.

A data event control block, shown in "Status information following an input/output operation" on page 373, is constructed as part of the macro expansion. If the OPEN macro specifies UPDAT, both the READ and WRITE macros must refer to the same data event control block. See the list form of the READ or WRITE macro for a description of how to construct a data event control block. See the execute form of the READ or WRITE macro for a description of how to modify an existing data event control block.

Data Conversion: For BSAM, you can request conversion by coding LABEL=(,AL) or (,AUL) in the DD statement, or by coding OPTCD=Q in the DCB macro or DCB subparameter of the DD statement. If conversion is requested, the check routine automatically converts BSAM records, as they are read, from one character representation to another if the record format is F, FB, D, DB, or U. Conversion occurs when the check routine determines that the input buffer is full. Conversion is performed according to one of the following techniques:

- **Coded Character Set Identifier (CCSID) Conversion**

  If CCSIDs are supplied from any source for ISO/ANSI V4 tapes, records are converted from the CCSID which represents the data on tape to the CCSID as seen by the problem program. You can also prevent conversion by supplying a special CCSID. CCSID may be supplied in the CCSID subparameter of a JOB, EXEC, or DD statement or the tape label.

- **Default Character Conversion**

  If you are using non-ISO/ANSI V4 tapes or if CCSIDs are not supplied by any source, data management converts the records from ASCII code to EBCDIC code using specific tables defined for this default character conversion.

Refer to *z/OS DFSMS Using Data Sets*, for a complete description of CCSID conversion and Default Character conversion.

Reading a PDSE directory: When reading the PDSE directory, end-of-file is indicated after the last of the directory data is read. Empty directory blocks are not simulated.

The system does not retain the record format (RECFM), logical record length (LRECL) and block size (BLKSIZE) when you write a z/OS UNIX file. If the defaults for reading are not appropriate, then you will have to supply them again when reading the file with BSAM, BPAM or QSAM. If you specify RECFM=V or VB, each READ returns one record per block. The system retains in the file directory entry the FILEDATA value that you code.

Extended format data sets: On READ requests for extended format data sets, that are not in the compressed format, you must provide a data area at least the size of DCBBLKSI or DCBEBLKSI unless you are reading format-U records and code a length on the READ macro. In that case, the data area must be at least the length coded.

Compressed format data sets: When processing a compressed format data set and NOTE/POINT is specified in the DCB (MACRF=P), a READ issued for a block whose user RBN value exceeds 16 777 215 will result in an I/O error. This is due to the fact that the NOTE/POINT interface is limited by a 3 byte token.

Addressing mode: When you issue the READ macro in 24-bit mode, provide only 24-bit addresses unless you code SF64 or SF64P. When you issue the READ macro in 31-bit addressing mode, provide only 31-bit addresses unless documentation says otherwise or you code SF64 or SF64P. With SF64 or SF64P, the data area can reside above the 2 GB bar but you cannot issue READ in 64-bit mode.

BSAM and BPAM allow data areas to be located above the 16MB line. This includes allowing the caller to issue some other BPAM and BSAM macros in 31-bit addressing mode regardless of whether the data area is above or below the 16MB line. Most types of data sets support 31-bit mode. See “Environmental considerations” on page xxii.

The standard form of READ must be issued from a program that resides below the 16MB line because the DECB must reside below the line.

To take advantage of providing data areas above the 16 MB line for BSAM macros, the issuer of the READ macro must execute in 31-bit addressing mode.

Syntax: The standard form of the READ macro is written as follows (the list and execute forms are shown following the descriptions of the standard form instructions):

[*label*]	READ	*decb name* ,{SF|SB|SF64|SF64P} ,*dcb address* ,*area address* [,*length*|,'S']

***decb name*—symbol**
specifies the name that is assigned to the data event control block (DECB) that is created as part of the macro expansion.

***type*—{SF|SB|SF64|SF64P}**
is coded as shown to specify the type of read operation:

SF
specifies normal, sequential, forward retrieval.

SB
specifies a read-backward operation. This parameter can be specified only for magnetic tape with format-F or format-U records.

This parameter is intended to be used when the data set is open for RDBACK. Tape positioning, label processing, and volume mounting errors will occur during EOV and CLOSE if an OPEN option other than RDBACK is used.

SF64
for BSAM, indicates sequential forward retrieval and that area address is an 8-byte address and can point to an area above the 2-GB bar. The 8-byte pointer in the macro expansion must reside

below the 2-GB bar. When the *area address* is in a register, it must be a 64-bit register. With SF64, you code the name or address of a double word that points to the area.

If you code SF64 on the list form (MF=L), then you must code SF64P on the execute form. That means that the execute form is providing a 64-bit pointer to the data area.

The system supports this option only for extended format data sets. For other restrictions see *z/OS DFSMS Using Data Sets*.

SF64P
for BSAM, indicates sequential forward retrieval and that *area address* is a doubleword (eight bytes) containing the address of the area above the 2-GB bar. The 8-byte pointer must reside below the 2-GB bar. With SF64, you code the name or address of the data area. With SF64P, you code the name or address of a doubleword that points to the area.

If you code SF64 on the list form (MF=L), then you must code SF64P on the execute form. That means that the execute form is providing a 64-bit pointer to the data area.

The system supports this option only for extended format data sets. For other restrictions see *z/OS DFSMS Using Data Sets*.

***dcb address*—A-Type Address or (2-12)**
specifies the address of the data control block for the opened data set to be read. When READ is issued in 31-bit addressing mode, the input DCB address and area address must be clean 31-bit addresses.

***area address*—A-Type Address or (2-12)**
specifies the address of the problem program area in which the block is placed if you do not code SF64P. Specifies an eight–byte pointer if you specify SF64P. If you specify SF64P, the specified doubleword contains an area pointer that can point above the 2 GB bar. The doubleword must reside below the 2 GB bar. If you specify the register form with SF64, it is a 64–bit register. If you specify SF64 or SF64P, your program must run in 24–bit or 31–bit mode.

***length*—symbol, decimal digit, absexp, (2-12), or 'S'**
specifies the number of data bytes to be read. This parameter is meaningful only for format-U records and for format-D records when the DCB BUFOFF parameter is not L. If your program is not using large block interface (LBI), the maximum value is the BLKSIZE value in the DCB when it was opened. If your program is using LBI, the maximum value is the BLKSIZE value in the DCBE.

For format-U records, 'S' or a valid length must be coded. If you code 'S', the number of bytes to be read is taken from the data control block BLKSIZE field (non-LBI) or the DCBE BLKSIZE field (LBI). For format-U records with LBI you must code 'S'.

For format-D records only, the length of the block just read is automatically inserted into the DCBLRECL field by the check routine if BUFOFF=L is not specified in the data control block.

READ—Read a block (offset read of keyed direct data set using BSAM)

The READ macro retrieves a block from a direct data set and places it in a designated area of storage. The data set is a direct, and its record format is unblocked variable-length spanned records. You must specify BFTEK=R in the data control block. Control might be returned to the problem program before the block is retrieved. The input operation must be tested for completion using a CHECK macro. A data event control block, shown in "Status information following an input/output operation" on page 373, is constructed as part of the macro expansion.

The standard form of the READ macro is written as follows (the list and execute forms are shown following the descriptions of the standard form):

[*label*]	READ	*decb name* ,SF ,*dcb address* ,*area address*

***decb name*—symbol**
specifies the name that is assigned to the data event control block (DECB) that is created as part of the macro expansion.

***type*—SF**
specifies normal, sequential, forward retrieval.

***dcb address*—A-Type Address or (2-12)**
specifies the address of the data control block for the opened direct data set to be read.

***area address*—A-Type Address or (2-12)**
specifies the address of the area in which the block is placed.

When a spanned direct data set is created with keys, only the first segment of a record has a key; successive segments do not. When a spanned record is retrieved by the READ macro, the system places a segment in a designated area addressed by the *area address*. The problem program must assemble all the segments into a logical record. Because only the first segment has a key, the successive segments are read into the designated area offset by key length to ensure that the block-descriptor word and the segment-descriptor word are always in their same relative positions.

READ—List and execute forms

READ—List form

The list form of the READ macro is used to construct a data management parameter list as a data event control block (DECB). For a description of the various fields of the DECB for each access method, see "Status information following an input/output operation" on page 373.

The description of the standard form of the READ macro explains the function of each parameter. The description of the standard form also indicates the parameters used for each access method, and the meaning of `'S'` when coded for the *area address*, *length*, and *key address* parameters. For each access method, `'S'` can be coded only for those parameters for which it can be coded in the standard form of the macro. The format description below indicates the optional and required parameters in the list form only.

The list form of the READ macro can be assembled into a program that resides above the 16MB line, but the execute form of the macro cannot use it there. You can copy it to below the 16MB line so the copy can be used, possibly in 31-bit mode.

The list form of the READ macro is:

[*label*]	READ	*decb name* ,*type* ,[*dcb address*] ,[*area address*|'S'] ,[*length*|'S'] ,[*key address*|'S'] ,[*block address*] ,[*next address*] ,MF=L

decb name—symbol

type—code one of the types shown in the standard form

dcb address—A-Type Address

area address—A-Type Address or 'S'

length—symbol, decimal digit, absexp, or 'S'

key address—A-Type Address or 'S'

block address—A-Type Address

next address—A-Type Address

MF=L
specifies that the READ macro is used to create a data event control block that can be referred to by an execute-form instruction.

READ—Execute form

A remote data management parameter list (data event control block) is used in, and can be modified by, the execute form of the READ macro. The data event control block can be generated by the list form of either a READ or WRITE macro.

The description of the standard form of the READ macro explains the function of each parameter. The description of the standard form also indicates the parameters used for each access method and the meaning of 'S' when coded for the *area address*, *length*, and *key address* parameters. For each access method, 'S' can be coded only for those parameters for which it can be coded in the standard form of the macro. The format description below indicates the optional and required parameters in the execute form only.

If your program executes in 31-bit mode, the execute form of READ may be issued above or below the 16MB line.

The execute form of the READ macro is:

[*label*]	READ	*decb address* ,*type* ,[*dcb address*] ,[*area address*|'S'] ,[*length*|'S'] ,[*key address*|'S'] ,[*block address*] ,[*next address*] ,MF=E

decb address—RX-Type Address or (1-12). This must reside below the 16MB line.

type—code one of the types shown in the standard form

dcb address—RX-Type Address or (2-12)

area address—RX-Type Address, (2-12), or 'S'

length—symbol, decimal digit, absexp, (2-12), or 'S'

key address—RX-Type Address, (2-12), or 'S'

block address—RX-Type Address, or (2-12)

next address—RX-Type Address or (2-12)

MF=E
specifies that the execute form of the READ macro is used, and that an existing data event control block (specified in the *decb address*) is used by the access method.

RELEX—Release exclusive control (BDAM)

The RELEX macro releases a data block from exclusive control. The block must have been requested in an earlier READ macro that specified either DIX or DKX.

Note: You can also use a WRITE macro that specifies either DIX or DKX to release exclusive control.

The RELEX macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

When the RELEX macro is issued in 31-bit addressing mode, the caller must ensure that the address of the input block reference field is a valid 31-bit address. It may reside above or below the line.

The format of the RELEX macro is:

[*label*]	RELEX	D ,*dcb address* ,*block address*

D
specifies direct access.

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for a direct data set opened for processing. The parameter must specify the same data control block designated in the associated READ macro.

***block address*—RX-Type Address, (2-12), or (0)**
specifies the address of the area containing the relative block address, relative track address, or actual device address of the data block being released. The parameter must specify the same area designated in the *block address* of the associated READ macro.

RELEX completion codes

When the system returns control to the problem program, the low-order byte of register 15 contains one of the following return codes. The 3 high-order bytes of register 15 are set to 0.

The RELEX return codes are:

Return Code (15)	Meaning
00 (X'00')	Operation completed successfully.
04 (X'04')	The specified data block was not in the exclusive control list.
08 (X'08')	The relative track address, relative block address, or actual device address was not in the data set.

RELSE—Release an input buffer (QISAM interface to VSAM and QSAM input)

The RELSE macro immediately releases the current input buffer. The next GET macro retrieves the first record from the next input buffer. For variable-length spanned records (QSAM), the input data set is spaced to the next segment that starts a logical record in a following block. Thus, one or more blocks of data or records might be skipped. The RELSE macro is ignored if a buffer has just been completed or released, if the records are unblocked, if it is issued for a SYSIN data set, or if it is issued for a UNIX file.

You can issue RELSE for QSAM in 24-bit mode or in 31-bit mode, but QISAM interface to VSAM requires 24-bit mode.

The format of the RELSE macro is:

[*label*]	RELSE	*dcb address*

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for the opened input data set. When issued in 31-bit addressing mode, the input DCB address must be a clean 31-bit address.

SETL—Set lower limit of sequential retrieval (QISAM interface to VSAM input)

The SETL macro causes the control program to start processing the next input request at the specified record or device address. Sequential retrieval of records using the GET macro continues from that point until the end of the data set is reached, or a CLOSE or ESETL macro is issued. You must issue an ESETL macro between SETL macros that specify the same data set.

The SETL macro can specify that retrieval is to start at the beginning of the data set, at a specific address on the device, at a specific record, or at the first record of a specific class of records. For additional information on SETL functions, see *z/OS DFSMS Using Data Sets*.

Recommendation: The system no longer supports indexed sequential data sets. Convert the data set to a key sequenced data set (KSDS) and use the ISAM interface of VSAM or convert your program to use VSAM.

The format of the SETL macro is:

[*label*]	SETL	*dcb address* {,K[H],*lower limit address*} {,KC,*lower limit address*} {,KD[H],*lower limit address*} {,KCD,*lower limit address*} {,I,*lower limit address*} {,ID,*lower limit address*} {,B} {,BD}

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block that is opened for the indexed sequential data set being processed.

The following parameters are coded as shown; they specify the starting point and type of retrieval:

K
specifies that the next input operation begins at the record containing the key specified in the *lower limit address*.

KC
specifies that the next input operation begins at the first record of the key class specified in the *lower limit address*. If the first record of the specified key class has been deleted, retrieval begins at the next non-deleted record regardless of key class.

H
used with either K or KD, specifies that, if the key in the *lower limit address* is not in the data set, retrieval begins at the next higher key. The character H cannot be coded with the key class parameters (KC and KCD).

KD
specifies that the next input operation begins at the record containing the key specified in the *lower limit address*, but only the data portion of the record is retrieved. This parameter is valid only for unblocked records.

KCD
specifies that the next input operation begins at the first record of the key class specified in the *lower limit address*, but only the data portion of the record is retrieved. This parameter is valid only for unblocked records.

I
specifies that the next input operation begins with the record at the actual device address specified in the *lower limit address*.

ID
specifies that the next input operation begins with the record at the actual device address specified in the *lower limit address*, but only the data portion of the record is retrieved. This parameter is valid only for unblocked records.

B
specifies that the next input operation begins with the first record in the data set.

BD
specifies that the next input operation begins with the first record in the data set, but only the data portion is retrieved. This parameter is valid only for unblocked records.

***lower limit address*—RX-Type Address, (2-12), or (0)**
specifies the address of the area containing the key, key class, or actual device address that designates the starting point for the next input operation. If I or ID is specified, the addressed area must contain the actual device address (in the form MBBCCHHR) of a prime data record; the other types require that the key or key class be contained in the addressed area.

SETL exit

The error analysis (SYNAD) routine is given control if the operation could not complete successfully. For information on how the exception condition code and general registers are set, see *z/OS DFSMS Using Data Sets*. If the SETL macro is not reissued, retrieval starts at the beginning of the data set.

SETPRT—Printer setup (BSAM, QSAM, and EXCP)

The SETPRT macro is used to initially set or dynamically change the printer control information for the IBM 3800 or 3900 Printing Subsystem and SYSOUT data sets.

Your program can issue the SETPRT macro in 31-bit or 24-bit, but the parameter list must reside below the 16 MB line.

3800 or 3900 printers and SYSOUT data sets

You can use SETPRT with any 3800 or 3900 model printer allocated to the program (not to JES). You can also use SETPRT when creating SYSOUT data sets. The SYSOUT data set does not have to be directed to an IBM 3800 or 3900 Subsystem or a printer. You can change the following control information with the SETPRT macro:

- Bursting of forms (BURST parameter).
- Character arrangements to be used (CHARS parameter).
- The number of copies (COPIES parameter).
- The starting copy number (COPYNR parameter).
- Vertical formatting of a page (FCB parameter).
- Flashing of forms (FLASH parameter).
- Initializing the printer control information (INIT parameter).
- Modification of copy (MODIFY parameter).
- Blocking or unblocking of data checks (OPTCD parameter).

Besides changing the control information, you can:

- Supply your own 3800 or 3900 load modules in a partitioned data set to replace the use of SYS1.IMAGELIB (LIBDCB parameter).
- SETPRT error messages that are sent to the printer can also be passed back to the invoking program (MSGAREA parameter).

- Print or suppress error messages on the directly allocated printer (PRTMSG parameter).
- Control the scheduling of SYSOUT segment printing (DISP parameter).

To use all-points addressability when operating the 3800 or 3900 Model 3, 6, or 8, use SYS1.FDEFLIB and SYS1.PDEFLIB instead of SYS1.IMAGELIB.

Not 3800 or 3900 printers

For printers other than the IBM 3800 or 3900 Printing Subsystem, SETPRT controls the following:

- Selection and verification of UCS and FCB images (UCS and FCB parameters).
- Blocking or unblocking of data checks (OPTCD parameter).
- Printing lowercase EBCDIC characters in uppercase (OPTCD and UCS parameters).
- Bypassing automatic forms positioning.

The SETPRT macro automatically positions forms in the printer to the first line of a new page when a new FCB is requested. If you wish to position the form yourself, specify the N option of the FCB parameter and insert the new form, matching the top of its page to the same position as the old form occupied.

This is how the SETPRT macro aligns a new form: If the FCB is different from the one currently in the printer, the old FCB and its current position is read from the printer. If the old form is not already at the top of a page, a temporary FCB is constructed and loaded back into the printer. A skip to 1 command is then executed to move the old form to the top of a new page. The requested FCB is then loaded into the printer. SETPRT's preparation is now complete. The new FCB and the old form are now at the first line of a new page. Printing is ready to start. If you wish to bypass automatic forms positioning, use the N option of the FCB parameter.

4248 printers

For the 4248 printer, the SETPRT macro controls following information:

- Activate, deactivate, and position horizontal copy (COPYP parameter).
- Speed of the printer (PSPEED parameter).

All supported devices

You can issue the SETPRT macro in 24-bit mode or 31-bit mode, but the standard and list forms and all modules to which the parameter list points must reside below the line.

When BSAM is used, all write operations must be checked for completion before the SETPRT macro is issued. Otherwise, an incomplete write operation might be purged.

Recommendations:

1. When processing a DCB that specifies QSAM locate mode and the buffers are above the 16MB line (DCBE RMODE31=BUFF is specified), issue the SETPRT macro in 31-bit mode.
2. A permanent error on a SETPRT macro causes one or both of the first two bits of the DCBIFLGS field to be set on. A cancel key or a paper jam that requires a printer subsystem-restart sets in the DCBIFLGS field the lost data-indicator bit, DCBIFLDT. After the system has turned on these bits, you must reset these bits to zero before you can reissue a SETPRT macro.

Unsupported devices

Issuing the SETPRT macro for a device other than a SYSOUT data set, a UCS printer, or the IBM 3800 or 3900 Printing Subsystem results in an error return code.

The standard form of the SETPRT macro is as follows (the list and execute forms are shown following the standard form):

[*label*]	SETPRT	*dcbaddr* [,BURST={N|Y}] [,CHARS={*name*|A(*address*)|R(*register*)}} {({*name*|A(*address*)| R(*register*)},...)}] [,COPIES=*number*] [,COPYNR=*number*] [,COPYP={*position*|0}] [,DISP={SCHEDULE|NOSCHEDULE|EXTERNAL}] [,FCB={*imageid*|A(*address*)|R(*register*)} ({*imageid*|A(*address*)| R(*register*)} [,{V|A}[,N]]) [,FLASH={NONE|*name*} *name*],*count*)}] [,INIT={N|Y}] [,LIBDCB=*dcb address*] [,MODIFY={{*name*|A(*address*)|R(*register*)} {({*name*|A(*address*)| R(*register*)},*trc*)}] [,MSGAREA=*address*] [,OPTCD={B|U} {({B|U},{F|U})}] [,PRTMSG={N|Y}] [,PSPEED={L|M|H|N}] [,REXMIT={N|Y}] [,UCS={*csc*} {(*csc*,{F|F,V|V})}]

***dcbaddr*—A-Type Address or (2-12)**
specifies the address of the data control block for the data set to be printed. The data set must be opened for output before the SETPRT macro is issued.

BURST={N|Y}
specifies whether the paper output is to be burst. BURST=Y indicates that the printed output is to be burst into separate sheets and stacked. BURST=N indicates that the printed output is to go into the continuous forms stacker. If BURST is not specified, the SETPRT routine assumes BURST=N. If bursting is requested, the printed output is threaded into the burster-trimmer-stacker. Otherwise, the printed output is threaded into the continuous forms stacker. The parameter prints a message at the system console telling the operator to thread the paper again if needed.

Restriction: This parameter is effective for the IBM 3800 or 3900 printer only.

CHARS={*name*|A(*address*)|R(*register*)}
{({*name*|A(*address*)|R(*register*) },...)}
specifies 1- to 4- character arrangement tables to be used when printing a data set.

Restriction: This parameter is effective for the IBM 3800 or 3900 printer only.

name
specifies the last four characters of the 8-byte member name for a character arrangement table module.

A(*address*)
specifies an in-storage address of the user-provided character arrangement table module. See *z/OS DFSMSdfp Utilities* for information on the format of the module.

R(*register*)
specifies the register containing an in-storage address of the user-provided character arrangement table module. For information on the format of the module, see *z/OS DFSMSdfp Utilities*.

COPIES=*number*
specifies the total number of copies of each page of the data set that is to be printed (from 1 to 255) before going to the next page. If COPIES is omitted, one copy of each page is printed.

Restriction: This parameter is effective for the IBM 3800 or 3900 printer only.

COPYNR=*number*
specifies the starting copy number for this transmission. *number* is a value from 1 to 255. This parameter defaults to a value of 1 if not specified.

Restriction: This parameter is effective for a directly-allocated IBM 3800 or 3900 printer only.

COPYP={*position*|0}
activates or deactivates the horizontal copy feature of the 4248 printer. This overrides the horizontal copy offset in the specified FCB. (If no FCB is specified, the horizontal copy offset in the already loaded FCB is overridden.) COPYP also controls horizontal copy capabilities with 3211 FCBs that are loaded in a 4248 printer.

position
specifies a decimal number from 2 to 168 indicating the print position where the horizontal copy starts. If your 4248 printer has only 132 print positions installed, the maximum number you should specify here is 132. When horizontal copy is activated, the maximum amount of data that can be sent to the printer is equal to the size of the smaller of the two copy areas. If the two copy areas are equal, the maximum amount of data that can be sent is equal to half the number of print positions.

For example, if you specify COPYP=101 for a 4248 printer with 132 print positions, the maximum amount of data that can be sent to the printer is 32 bytes. (Thirty-two bytes is equal to the smaller copy area, from position 101 to position 132.) If you specify COPYP=67 for a 4248 printer with 132 print positions, the maximum amount of data that can be printed is 66 bytes. (Sixty-six bytes is equal to half the number of print positions.)

If COPYP=*position* is specified and a 3211 format FCB is being used, the 3211 format FCB is converted to 4248 format FCB and the specified offset value is inserted.

Restriction: COPYP=position is not available with the IBM 3262 Model 5 printer.

0
specifies that no horizontal copy is to be made. Any offset value in the specified or already loaded FCB is overridden.

Rule: Channel programs that are used when horizontal copy is activated *must* have the suppress length indication (SLI) bit set.

DISP={SCHEDULE|NOSCHEDULE|EXTERNAL}
DISP allows you to control how JES disposes of the data created before the SETPRT request. This parameter is valid only for SYSOUT data sets and is ignored for the direct user who issues SETPRT. You can abbreviate the parameters to S, N, and E, respectively. This parameter is effective for any SYSOUT data set.

SCHEDULE
specifies that JES is to schedule the previous data for printing immediately.

NOSCHEDULE
specifies that JES is to separate the data into a separate JES data set and to schedule the previous data set for printing after the job terminates.

EXTERNAL
specifies that the schedule of the data set for printing is determined by the JCL parameter FREE=CLOSE. FREE=CLOSE is the same as specifying DISP=SCHEDULE. The absence of FREE=CLOSE in the JCL is the same as coding DISP=NOSCHEDULE on the SETPRT macro. EXTERNAL is the default.

FCB={*imageid*|A(*address*)|R(*register*)}
({*imageid*|A(*address*)|R(*register*)}[,{V|A}[,N])
specifies that the forms control buffer (FCB) is selected from the image library. The possible specifications are:

imageid
specifies the forms control image to be loaded. A forms control image is identified by a 1- to 4-character name. IBM-supplied 3211 format images are identified by imageid value of STD1 and STD2. User-designed forms control images are defined by the installation.Note that the 4248 accepts both 3211 and 4248 format FCBs. For descriptions of the standard forms control images for the 3203 and 3211, 3262 Model 5 or 4245, see *z/OS DFSMSdfp Advanced Services*. For a description of the 4248 FCB, see *z/OS DFSMSdfp Utilities*. For more information about 3800 or 3900 FCB modules, see *z/OS DFSMSdfp Utilities*.

A(*address*)
specifies an in-storage address of the user-supplied forms control buffer module to be used. (For information on the format of the module, see *z/OS DFSMSdfp Utilities*.)

Restriction: This subparameter is effective for directly-allocated IBM 3800 or 3900 Model 1 printers.

R(*register*)
specifies the register that contains an in-storage address of the user-provided forms control buffer module to be used when printing a data set. (For information on the format of the module, see *z/OS DFSMSdfp Utilities*.)

Restriction: This subparameter is effective for directly-allocated IBM 3800 or 3900 Model 1 printers.

V or VERIFY
requests that the forms control image be displayed on the printer for visual verification. This subparameter allows forms verification and alignment using the WTOR macro.

A or ALIGN
allows forms alignment using the WTOR macro. This subparameter is ignored if specified for the IBM 3800 or 3900 printer.

N
bypasses automatic forms positioning. This subparameter is ignored if specified for the IBM 3800 or 3900 printer. N is not available via JCL and, thus, cannot be used when opening a directly-allocated printer because OPEN obtains printer setup parameters from the JCL.

FLASH={NONE|*name*|}

([*name*],*count*)}
identifies the forms overlay frame to be used. Unless REXMIT=Y is coded and the forms overlay frame is still in use from the previous SETPRT macro issuance, a message tells the operator to insert this forms overlay frame into the printer. This parameter also lets you specify the number of copies on which the overlay is to be printed (flashed). If you omit this parameter for a directly attached printer, flashing stops. If you omit this parameter when doing a SETPRT while generating SYSOUT data, the FLASH parameters previously in effect for this data set are used.

Restriction: This parameter is effective for the IBM 3800 or 3900 printer only.

NONE
is valid only when using SETPRT while generating SYSOUT data, and causes zero copies to be flashed. If flashing is resumed in a later SETPRT, a message is generated by JES regarding the insertion of the forms overlay frame, even if no change in the forms overlay frame is necessary.

name
specifies the 1- to 4-character name of the forms overlay frame.

count
specifies the total number (0 to 255) of copies of each page of the data set on which the overlay is to be printed, beginning with the first copy. The number of copies printed is not greater than the number of copies specified by COPIES.

For a directly attached printer: No copies are flashed if you specify a flash count of zero. If you specify a nonzero flash count and omit the name of the forms overlay frame, the operator is not requested to insert a frame. Whatever frame is inserted is used.

During the generation of SYSOUT data: If you specify a flash count of zero, the flash count previously in effect for the data set is used. If you specify a nonzero flash count and omit the name of the forms overlay frame, the operator is not requested to insert a frame except when flashing has stopped. If flashing stops, a message from JES requests the operator to insert a new frame. Then, the flashing of the forms resumes using the count specified in the flash count parameter.

INIT={N|Y}
When INIT=Y is specified for a directly-allocated IBM 3800 printer, it initializes the control information in the printer with a folded character arrangement table: the 10-pitch Gothic character set (12 pitch for the IBM 3800 or 3900 Models 3, 6, and 8), and a 6 lines per inch FCB corresponding to the forms size in the printer. COPIES and COPYNR are initialized to 1, FLASH and MODIFY are cleared, and BURST is initialized to N (continuous forms).

When INIT=Y is specified for a SYSOUT data set, other parameters not specified on the same invocation are reset, meaning the JES default is used. ("JES default" refers to what was specified when JES was set up.) For INIT=N, all control information for the IBM 3800 or 3900 printer remains unchanged. Any parameters included on the same macro statement as INIT are processed after printer initialization completes.

Restriction: This parameter is effective for the IBM 3800 or 3900 printer only.

LIBDCB=*dcb address*—A-Type Address or (2-12)
dcb address is the address of an authorized user library DCB that has been opened, and that you want to use instead of SYS1.IMAGELIB. If LIBDCB is not specified, SYS1.IMAGELIB is used.

Restriction: This parameter is effective for directly-allocated IBM 3800 or 3900 printers only.

MODIFY={*name*|A(*address*)|R(*register*)}
{({*name*|A(*address*)| R(*register*)},*trc*)}
identifies the copy modification module and an associated character arrangement table module used when modifying the data to be printed.

Restriction: This parameter is effective for IBM 3800 or 3900 printers or SYSOUT.

name
specifies the 1- to 4-character name of the copy modification module stored in SYS1.IMAGELIB. These one to four characters are the fifth to eighth characters of the 8-byte member name of a copy modification module in SYS1.IMAGELIB.

A(*address*)
specifies an in-storage address of the user-supplied copy modification module. For information on the format of the module, see *z/OS DFSMSdfp Utilities*.

Restriction: This subparameter is effective for the IBM 3800 or 3900 Model 1 printer.

R(*register*)
specifies the register containing an in-storage address of the user-provided copy modification module. For information on the format of the module, see *z/OS DFSMSdfp Utilities*. This subparameter is effective for the IBM 3800 or 3900 Model 1 printer.

trc
specifies the table reference character used to select one of the character arrangement table modules to be used for the copy modification text. The values of 0, 1, 2, and 3 correspond to the

order in which the module names are specified in CHARS. If *trc* is not included, the first character arrangement table module (0) is assumed.

MSGAREA=*address*—A-Type Address or (2-12)
address is the address of the message feedback area. This area is used to transfer message text between the SETPRT macro and the caller. You must allow at least 80 bytes for the message text plus 10 bytes for prefix information or a total length of at least 95 bytes. The message is truncated if it does not fit into the area. This area resides below the 16MB line.

Restriction: This parameter is effective for the IBM 3800 or 3900 printer only.

The following shows the layout of the message area:

bytes 0-1:
total length

bytes 2-5:
reserved

bytes 6-7:
text length

bytes 8-9:
reserved

bytes 10-variable:
message text

OPTCD={B|U}
{({B|U},{F|U})}
specifies whether printer data checks are blocked or unblocked and if the printer is to operate in fold or normal mode. You can specify:

B
specifies that printer data checks are blocked (this means to suppress them). This option updates the DCBOPTCD field of the data control block.

U
specifies that printer data checks are unblocked. This option updates the DCBOPTCD field of the data control block.

FOLD or F
specifies that printing is in fold mode. This subparameter is ignored if specified for the IBM 1403 or IBM 3800 or 3900 printer. For 1403 fold mode, use FOLD option under the UCS parameter.

UNFOLD or U
specifies that printing is in normal mode. This subparameter causes fold mode to revert to normal mode. This subparameter is ignored if specified for the IBM 1403 or IBM 3800 or 3900 printer. Because UCS processing occurs after OPTCD processing, if FOLD is specified in the UCS parameter, fold mode is set. If FOLD is not coded, unfold is set.

PRTMSG={N|Y}
allows printing of printer error messages for the programmer on the IBM 3800 or 3900. This parameter is effective for the 3800 only.

N
specifies not to print error messages on the IBM 3800 or 3900.

Y
specifies to print error messages on the IBM 3800 or 3900. Y is the default.

PSPEED={L|M|H|N}
specifies the printer's speed, which affects print quality. This parameter is effective for the 4248 printer only, and is ignored for all other printers. LOW speed produces the best quality. PSPEED is used to set the printer's speed or override that set in the FCB. If no FCB is specified, the PSPEED parameter, if any, in the already loaded FCB is used.

L or LOW
sets the printer speed to 2200 lines per minute.

M or MEDIUM
sets the printer speed to 3000 lines per minute.

H or HIGH
sets the printer speed to 3600 lines per minute.

N or NOCHANGE
indicates that the speed at which the printer is currently running is to remain the same no matter what is specified in the requested FCB, or if none is specified, in the already loaded FCB.

Actual printer speed can vary.

REXMIT={N|Y}
specify REXMIT=Y to modify the starting copy number (COPYNR), the number of copies of the pages in a data set to be printed (COPIES), the forms overlay frame to be used (FLASH), and the number of copies to be printed (FLASH) without changing the other control information already set up in the printer. The SETPRT SVC ignores all other parameters in the parameter list.

UCS={*csc*}
{(*csc*,{F|F,V|V})}
specifies that the character set image that is to be used. This parameter is ignored if specified for the IBM 3800 or 3900 printer. You can specify:

***csc* (character set code)**
specifies the character set selected. A character set is identified by a 1- to 4-character code. Codes for standard IBM character sets are as follows:

1403 or 3203 Printer: AN, HN, PCAN, PCHN, PN, QN, QNC, RN, SN, TN, XN, and YN

3211 Printer: A11, H11, G11, P11, and T11

4245 Printer: AN21, AN31, HN21, HN31, PL21, PL31, GN21, RN21, RN31, TN21, SN21, FC21, KA21, and KA22

4248 Printer: 40E1, 40E2, 4101, 4102, 4121, 4122, 41C1, 41C2, 4181, 4201, 4061, 40C1, 4161, 4041, and 4042

Note: There are no standard IBM character sets supplied for the IBM 3262 Model 5 printer.

The 4245 and 4248 printers load their own images on recognition of the mounted band. The image table provides a correspondence between the band identification and the character set code.

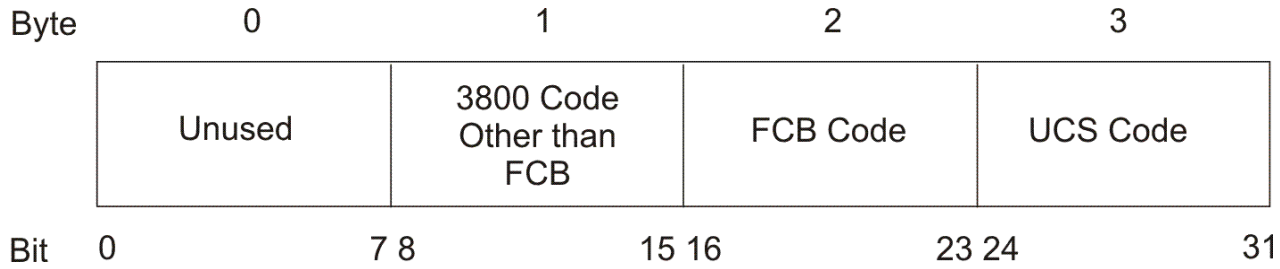
See *z/OS DFSMSdfp Advanced Services* for a description of the 4245 and 4248 UCS image tables and information on adding user-defined entries to an image table.

FOLD or F
specifies that the character set image that is selected be in fold mode. The fold mode converts the EBCDIC code for lowercase characters to the EBCDIC code for the corresponding uppercase characters. Unless FOLD is specified, UNFOLD mode is set.

V or VERIFY
requests that the character set image be displayed on the printer for visual verification.

SETPRT return codes

After the SETPRT macro is executed, a return code is placed in register 15, and control is returned to the instruction following the SETPRT macro. The illustration below shows how the 4 bytes of register 15 are used for a specific printer.



Return codes X'0' through X'24' apply to all printers.

Return codes X'28' through X'4C' apply to the 3800 or 3900 printer only. There is one exception; return code X'48' also applies to the IBM 3262 Model 5 and the IBM 4248 printer.

Return code X'50' applies to SYSOUT data sets.

Return codes 0 to 14

Table 48 on page 323 shows the hexadecimal return codes X'00' through X'14' for specific printers.

Table 48. SETPRT Return Codes 00 to 14

3800 or 3900 Code Other than FCB (Byte 1)	FCB Code (Byte 2)	UCS Code (Byte 3)	Meaning
00	00	00	Successful completion.
00	00	04	The operator canceled the UCS request for one of the following reasons: • The UCS image could not be found in SYS1.IMAGELIB. • The requested train or band was not available.
00	04	00	For not 3800 or 3900 printers, the operator canceled the FCB load operation for one of the following reasons: • The form could not be aligned to match the buffer. • The FCB module could not be found in SYS1.IMAGELIB or your DCB exit list. For a 3800 or 3900, the specified FCB module could not be found in SYS1.IMAGELIB, a user library, or the DCB exit list, and SETPRT processing was terminated.

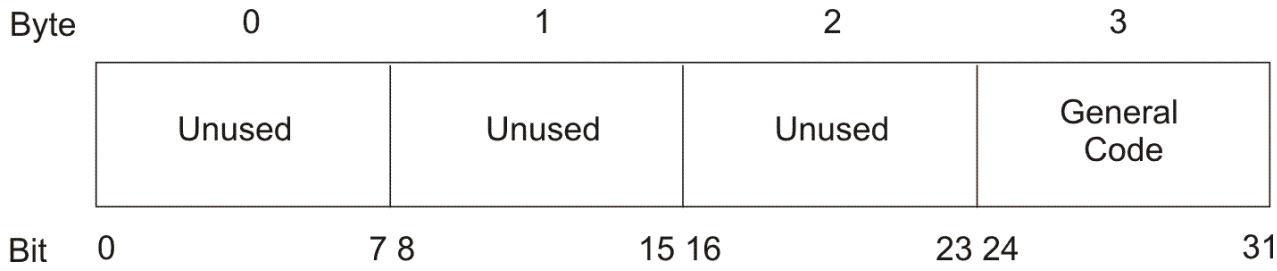
Table 48. SETPRT Return Codes 00 to 14 (continued)

3800 or 3900 Code Other than FCB (Byte 1)	FCB Code (Byte 2)	UCS Code (Byte 3)	Meaning
04	00	00	The 3800 or 3900 SETPRT processing was suspended for one of the following reasons: • A character arrangement table module could not be found in SYS1.IMAGELIB or a user library. • A copy modification module could not be found in SYS1.IMAGELIB or a user library. • A graphic character modification module (required by a character arrangement table module) could not be found in SYS1.IMAGELIB or a user library. • A library character set module could not be found in SYS1.IMAGELIB or a user library. Register 0 contains a reason code identifying which of the above conditions occurred. For an explanation, see Table 50 on page 328.
00	00	08	A permanent I/O error was detected when the BLDL macro was issued to locate a UCS image or image table in SYS1.IMAGELIB.
00	08	00	A permanent I/O error was detected when the BLDL macro was issued to locate an FCB module in SYS1.IMAGELIB or a user library.
08	00	00	A permanent I/O error was detected when the BLDL macro was issued to locate one of the following modules in SYS1.IMAGELIB or a user library. • A character arrangement table module. • A copy modification module. • A graphic character modification module. • A library character set module. Register 0 contains a reason code identifying which of the above conditions occurred. For an explanation, see Table 50 on page 328.
00	00	0C	A permanent I/O error was detected while loading the printer's UCS buffer, or displaying a message on the 4248 printer.
00	0C	00	A permanent I/O error was detected during forms positioning or while loading the printer's FCB buffer. Register 0 contains a reason code identifying which of the above conditions occurred. For an explanation, see Table 54 on page 329.

Table 48. SETPRT Return Codes 00 to 14 (continued)

3800 or 3900 Code Other than FCB (Byte 1)	FCB Code (Byte 2)	UCS Code (Byte 3)	Meaning
0C	00	00	A permanent I/O error was detected while loading one of the following: • Character arrangement table. • Copy modification record. • Starting copy number. • Graphic character modification record. • Forms overlay sequence control record (copy counts and flash counts). • Writable character generation module (WCGM). • Library character set. Register 0 contains a reason code identifying which of the above conditions occurred. For an explanation, see Table 50 on page 328.
00	00	10	A permanent I/O error was detected during UCS verification display or while reading the UCS buffer.
00	10	00	A permanent I/O error was detected during FCB verification display.
00	00	14	The operator canceled the UCS request because an improper character set image was displayed for visual verification.
00	14	00	The operator canceled the FCB request because an improper forms control image was displayed for visual verification.

The illustration below shows how the 4 bytes of register 15 are used for all printers.



Return codes 18 to 50

Table 49 on page 326 shows the return codes X'18' through X'50' for all printers.

Table 49. SETPRT Return Codes 18 to 50

Return Code (Byte 3)	Meaning
X'18'	No operation was performed for one of the following reasons: • The data control block was not open. • The data control block was not valid for a sequential data set. • The SETPRT parameter list was not valid. • The output device was not a UCS or 3800 or 3900 printer or SYSOUT. • SETPRT was issued to a UNIX file.
X'1C'	No operation was performed because an uncorrectable error occurred in a previously initiated output operation. The error analysis (SYNAD) routine is entered when the next PUT or CHECK macro is issued. No operation was performed because an uncorrectable error occurred when the block data check or the reset block data check command was issued by SETPRT. For a 4245, a possible lost data condition was detected. For a 3800 or 3900, message IEC173I indicates which of the above errors has occurred. Register 0 contains a reason code identifying whether data was lost.
X'20'	Not enough storage was available for opening the SYS1.IMAGELIB, or, for a 3800 or 3900 printer, not enough storage was available to contain the control blocks for a user library, or insufficient storage was available for SETPRT.
X'24'	SYS1.IMAGELIB (or, for the 3800 or 3900 printer, a user library) cannot be opened to load the specified module. Either: • a permanent I/O error occurred • SYS1.IMAGELIB was mounted or cataloged incorrectly, • SYS1.IMAGELIB is an alias for a data set for which you do not have RACF read authority.
X'28'	The operator canceled the forms overlay request.
X'2C'	The operator canceled the paper threading request.
X'30'	More writable character generation modules (WCGMs) were requested than there are writable buffers installed on the printer.
X'34'	There was an invalid table reference character for copy modification module.
X'38'	An error occurred when attempting to execute the initialize printer command.
X'3C'	Bursting was requested but the burster-trimmer-stacker feature is not installed on the printer.
X'40'	A permanent I/O error occurred while executing a sense, final select character arrangement table command, or display status code.
X'44'	The translate table character arrangement table entry references a character set that is not in the image library.

Table 49. SETPRT Return Codes 18 to 50 (continued)

Return Code (Byte 3)	Meaning
X'48'	Data was lost because of one of the following (3800 or 3900 only): • 3800 or 3900 system restart after a paper jam. • Cancel key. • Lost resources after paper jam. For a 4248, a possible lost data condition was detected. Register 0 contains a reason code identifying which of the above conditions occurred. See Table 52 on page 328 for an explanation.
X'4C'	A load check was detected while loading one of the following (3800 or 3900 only): • Forms control buffer (FCB). • Character arrangement table (CAT). • Graphic arrangement table (GCM). • Copy modification record. • Writable character generation module (WCGM). • Library character set (LCS). Register 0 contains a reason code identifying which of the above conditions occurred. For an explanation, see Table 50 on page 328.
X'50'	When a SETPRT was issued to a direct attach (an online 3800 or 3900 Model 3, 6 or 8 printer) or a SYSOUT data set, there was a failure in one of the following: • OPEN or CLOSE. • Data set segmentation. • Processing of system control blocks. • Obtaining exclusive control. • More than one DCB is open for the SYSOUT data set. For an explanation of the reason codes associated with return code 50, see Table 53 on page 328.

SETPRT reason codes

All 3800 or 3900 printers

The following illustration shows the contents of register 0, which includes the GCM ID, the CAT ID, and the reason code.

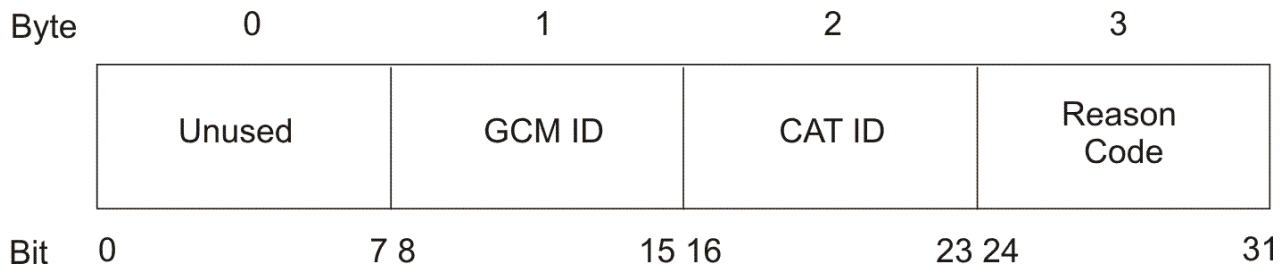


Table 50 on page 328 shows the hexadecimal reason codes for the IBM 3800 Model 1 and the other 3800 or 3900 models in compatibility mode. These reason codes, returned in register 0, are in addition to return codes X'04', X'08', X'0C', and X'4C' returned in register 15.

Table 50. Reason Codes for IBM 3800 or 3900 Printers (for Return Codes 04, 08, 0C, 4C)

GCM ID (Byte 1)	CAT ID (Byte 2)	Reason Code (Byte 3)	Meaning
00	01-04	04	Character arrangement table module/record.
00	00	08	Copy modification module/record.
00	00	0C	Starting copy number.
01-04	01-04	10	Graphic character modification module/record.
00	00	14	Forms overlay sequence control record.
00	00	18	Library character set.
00	00	1C	Writable character generation module (WCGM).
00	00	20	Forms control buffer module.

3800 or 3900 printers and the 4245 printer

These reason codes (Table 51 on page 328) apply to all 3800 or 3900 printers and the IBM 4245 printer. Return code X'1C' returned in register 15. The reason code is placed in byte 3 of register 0.

Table 51. Reason Codes for All Printers (for Return Code 1C)

Reason Code (Byte 3)	Meaning
X'00'	Indicates no data lost.
X'04'	Indicates data has been lost.

Table 52 on page 328 shows the reason codes in addition to return code X'48' returned in register 15. The reason code is placed in byte 3 of register 0.

Table 52. Reason Codes for 3800 or 3900 Printers and 4248 Printer (for Return Code 48)

Reason Code (Byte 3)	Meaning
X'04'	A paper jam caused a restart. A possible lost data condition was detected.
X'08'	The cancel key was pressed.
X'0C'	Resources were lost after a paper jam.

Table 53 on page 328 shows the reason codes in addition to return code X'50' returned in register 15. The reason code is placed in byte 3 of register 0.

Table 53. Reason Codes for Return Code 50

Reason Code (Byte 3)	Meaning
X'04'	An invalid SETPRT request for a SYSOUT data segment was specified. An in-storage address was used for a copy modification, character arrangement table, FCB, or user library DCB. Only load module IDs in SYS1.IMAGELIB are allowed for SYSOUT setup.

Table 53. Reason Codes for Return Code 50 (continued)

Reason Code (Byte 3)	Meaning
X'08'	During SETPRT processing for a SYSOUT data segment, an error was detected while attempting to read a JFCB or JFCBE control block from SWA.
X'0C'	During SETPRT processing for a SYSOUT data segment, an error was detected while invoking the CLOSE subsystem interface (SSI) for the previous data segment.
X'10'	During SETPRT processing for a SYSOUT data segment, an error was detected while invoking the OPEN subsystem interface (SSI) for the new data segment being created.
X'14'	During SETPRT processing for a SYSOUT data segment, an error was detected while the scheduler spool file allocation routine was segmenting the data set.
X'18'	An ENQ macro failed. The ENQ was issued by SETPRT processing.
X'1C'	More than one DCB is open for the SYSOUT data set.

All not 3800 or 3900 printers

Table 54 on page 329 shows the reason code in addition to completion code 0C00.

Table 54. Reason Codes for Not 3800 or 3900 Printers (for Completion Code 0C00)

Reason Code (Byte 3)	Meaning
X'00'	The I/O error was not caused by a load check.
X'04'	FCB load failed because of a load check. Probably caused by invalid FCB contents.

SETPRT—List form

The list form of the SETPRT macro is used to construct a data management parameter list. The description of the standard form of the SETPRT macro explains the function of each parameter. The *dcbaddr* must appear in the list or execute form of the SETPRT macro.

The parameter list must reside below the 16MB line.

The list form of the SETPRT macro is as follows:

[*label*]	SETPRT	[*dcbaddr*] [,BURST={N|Y}] [,CHARS={[*name*} {(*name*,...)}] [,COPIES=*number*] [,COPYNR=*number*] [,COPYP={*position*|0}] [,DISP={SCHEDULE|NOSCHEDULE|EXTERNAL}] [,FCB={*imageid*} (*imageid*,{V|A}[,N]) [,FLASH={NONE|*name*} {NONE|([*name*],*count*)}] [,INIT={N|Y}] [,LIBDCB=*dcb address*] [,MODIFY={*name*} {(*name*,*trc*)}] [,MSGAREA=*address*] [,OPTCD={B|U} {({B|U},{F|U})}] [,PRTMSG={N|Y}] [,PSPEED={L|M|H|N}] [,REXMIT={N|Y}] [,UCS={*csc*} {(*csc*,{F|F,V|V})}] ,MF=L

dcbaddr—A-Type Address

BURST={N|Y}
is coded as shown in the standard form of the macro.

CHARS={*name*}
{(*name*,...)}
is coded as shown in the standard form of the macro, except for the A(*address*) and R(*register*) parameters, which cannot be specified.

COPIES=*number*
is coded as shown in the standard form of the macro.

COPYNR=*number*
is coded as shown in the standard form of the macro.

COPYP={*position*|0}
is coded as shown in the standard form of the macro.

DISP={SCHEDULE|NOSCHEDULE|EXTERNAL}
is coded as shown in the standard form of the macro.

FCB={*imageid*}
(*imageid*,{V|A}[,N])
is coded as shown in the standard form of the macro, except for the A(*address*) and R(*register*) parameters, which cannot be specified.

FLASH={NONE|*name*}
{([*name*],*count*)}
is coded as shown in the standard form of the macro.

INIT={N|Y}
is coded as shown in the standard form of the macro.

LIBDCB=*dcb address*—RX-Type Address or (2-12)
is coded as shown in the standard form of the macro.

MODIFY={*name*}
{(*name*,*trc*)}
is coded as shown in the standard form of the macro, except for the A(*address*) andR(*register*) parameters, which cannot be specified.

MSGAREA=*address*—RX-Type Address or (2-12)
is coded as shown in the standard form of the macro.

OPTCD={B|U}
{({B|U},{F|U})}
is coded as shown in the standard form of the macro.

PRTMSG={N|Y}
is coded as shown in the standard form of the macro.

PSPEED={L|M|H|N}
is coded as shown in the standard form of the macro.

REXMIT={N|Y}
is coded as shown in the standard form of the macro.

UCS={*csc*}
{(*csc*,{F|F,V|V})}
is coded as shown in the standard form of the macro.

MF=L
specifies that the list form of the macro is used to create a parameter list that can be referred to by an execute form of the SETPRT macro.

SETPRT—Execute form

A remote data management parameter list is referred to, and can be modified by, the execute form of the SETPRT macro.

The description of the standard form of the SETPRT macro explains the function of each parameter. The *dcbaddr* must be specified in the list or execute form of the SETPRT macro.

The execute form of the SETPRT macro is as follows:

[*label*]	SETPRT	[*dcbaddr*] [,BURST={N|Y|*}] [,CHARS={*name*|A(*address*)|R(*register*)} {({*name*|A(*address*)|R(*register*)},...)} {*}] [,COPIES={*number*|*}] [,COPYNR={*number*|*}] [,COPYP={*position*|0}] [,DISP={SCHEDULE|NOSCHEDULE|EXTERNAL}] [,FCB={*imageid*|A(*address*)|R(*register*)} ({*imageid*|A(*address*)|R(*register*)[,{V|A} [,N]]) {*}] [,FLASH={NONE|*name*} {([NONE|*name*],*count*)} {*}] [,INIT={N|Y}] [,LIBDCB=*dcb address*] [,MODIFY={*name*|A(*address*)|R(*register*)*} {({*name*|A(*address*)|R(*register*)},*trc*)} {*}] [,MSGAREA=*address*] [,OPTCD={B|U} {({B|U},{F|U})}] [,PRTMSG={N|Y}] [,PSPEED={L|M|H|N}] [,REXMIT={N|Y|*}] [,UCS={*csc*} {(*csc*,{F|F,V|V})}] ,MF=(E,*data management list address*)

dcbaddr—RX-Type Address or (2-12)

BURST={N|Y|*}
is coded as shown in the standard form of the macro, except for the * subparameter, which can be used only when INIT=Y is specified in the execute form of the SETPRT macro. When BURST=* is coded, the BURST field in the parameter list remains as previously set. This parameter is effective for the IBM 3800 or 3900 printer only.

CHARS={*name*|A(*address*)|R(*register*)}
{({*name*|A(*address*)|R(*register*)},...)}
{*}
is coded as shown in the standard form of the macro, except for the * subparameter, which can be used only when INIT=Y is specified in the execute form of the SETPRT macro. When CHARS=* is coded, the CHARS field in the parameter list remains as previously set.

COPIES={*number*|*}
is coded as shown in the standard form of the macro, except for the * subparameter, which can be used only when INIT=Y is specified in the execute form of the SETPRT macro. When COPIES=* is coded, the COPIES field in the parameter list remains as previously set.

COPYNR={*number*|*}
is coded as shown in the standard form of the macro, except for the * subparameter, which can be used only when INIT=Y is specified in the execute form of the SETPRT macro. When COPYNR=* is coded, the COPYNR field in the parameter list remains as previously set.

COPYP={*position*|0}
is coded as shown in the standard form of the macro.

DISP={SCHEDULE|NOSCHEDULE|EXTERNAL}
is coded as shown in the standard form of the macro.

FCB={*imageid*|A(*address*)|R(*register*)}
({*imageid*|A(*address*)|R(*register*)}[,{V|A}[,N])
{*}
is coded as shown in the standard form of the macro, except for the * subparameter, which can be used only when INIT=Y is specified in the execute form of the SETPRT macro. When FCB=* is coded, the FCB field in the parameter list remains as previously set.

FLASH={NONE|*name*}
{NONE|([*name*],*count*)}
{*}
is coded as shown in the standard form of the macro, except for the * subparameter, which can be used only when INIT=Y is specified in the execute form of the SETPRT macro. When FLASH=* is coded, the FLASH field in the parameter list remains as previously set.

INIT={N|Y}
is coded as shown in the standard form of the macro. When INIT=Y is specified on the execute form of the SETPRT macro, all 3800 or 3900 fields in the parameter list (BURST, CHARS, COPIES, COPYNR, FCB, FLASH, MODIFY, and REXMIT) are reset to binary zeros unless a specified field is preserved by coding keyword parameter=* or changed by specifying a valid subparameter for the keyword parameter as described in the standard form of the macro.

LIBDCB=*dcb address*—A-Type Address or (2-12)
is coded as shown in the standard form of the macro.

MODIFY={*name*|A(*address*)|R(*register*)}
{({*name*|A(*address*)|R (*register*)},*trc*)}
{*}
is coded as shown in the standard form of the macro, except for the * subparameter, which can be used only when INIT=Y is specified in the execute form of the SETPRT macro. When MODIFY=* is coded, the MODIFY field in the parameter list remains as previously set.

MSGAREA=*address*—A-Type Address or (2-12)
is coded as shown in the standard form of the macro.

OPTCD={B|U}
{({B|U},{F|U})}
is coded as shown in the standard form of the macro.

PRTMSG={N|Y}
is coded as shown in the standard form of the macro.

PSPEED={L|M|H|N}
is coded as shown in the standard form of the macro.

REXMIT={N|Y|*}
is coded as shown in the standard form of the macro, except for the * subparameter, which can be used only when INIT=Y is specified in the execute form of the SETPRT macro. When REXMIT=* is coded, the REXMIT field in the parameter list remains as previously set.

UCS={*csc*}
{(*csc*,{F|F,V|V})}
is coded as shown in the standard form of the macro.

MF=(E,*data management list address*)
specifies that the execute form of the SETPRT macro is used, and that an existing data management parameter list is used.

E

data management list address—RX-Type Address, (2-12), or (1).

STOW—Update partitioned data set directory (BPAM)

The STOW macro updates a partitioned data set (PDS) directory or PDSE directory. You can perform the following directory actions with STOW:

- Add a new member or alias name.
- Replace a member or alias name. The member does not need to exist in the directory.
- Change the name of an existing member or alias.
- Delete an existing member or alias name.
- Initialize (or reset to empty) the directory of a PDS or PDSE.
- Disconnect PDSE members.
- Replace a member of a PDSE, if and only if the existing member was created with a specified timestamp value.
- Replace a generation of a member in a PDSE. The generation does not need to exist in the PDSE.
- Delete a generation of a member in a PDSE.
- Recover (or make current) a generation of a member in a PDSE.

Your program can issue the STOW macro in 31-bit or 24-bit, but the parameter list must reside below the 16 MB line.

The format of the STOW macro is:

[*label*]	STOW	*dcb address* ,*list address* [,*directory action*]

***dcb address*—RX-Type Address, (2-12), or (1)**
Specifies the address of an open DCB.

***directory action*—[A|C|D|I|DISC|IFF|RG|DG|RECOVERG]**
If *directory action* is not coded, A(add an entry) is the default. The parameter is coded as shown to specify the type of directory action. You can issue IFF, RG, DG, and RECOVERG against a PDSE only.

A
Specifies that the name of a new member or alias is to be added to the directory.

C
Specifies that the name of an existing member or alias is to be changed. For a PDSE which supports generations, any prior generations remain associated with the old name.

D
Specifies that the current directory entry for a member is to be deleted. For PDSEs, when the member name is deleted, all aliases for that member are deleted. For PDSEs which support generations, only the current member is deleted. The deleted member is retained as the newest generation and the oldest generation might be permanently deleted if it exceeds the maximum number of generations defined for the PDSE.

I
Initializes, or resets to empty, a PDS or PDSE directory. The parameter list (list address) is not required for STOW initialize. There are no serialization requirements for using STOW initialize other than the PDS must not be open for writing with any other DCB. For PDSEs, member connections protect an application that is accessing a member concurrent to the issuance of STOW initialize. A member being read will be marked for deletion, but will not be deleted until after the connection is released.

For a PDSE with generations, all generations are deleted.

For a PDS, the next write will be to where the oldest member used to be after the directory. When you close the DCB, the system updates the DSCB to show the new last block in the data set. This means that there will be no benefit to attempting to compress the PDS.

R
Specifies that an existing member or alias directory entry is to be replaced by a new directory entry. If the old entry does not exist, the new entry is added to the directory and a completion

code of X'08' is returned in register 15. For PDSEs, when the member name is replaced, all aliases for that member are deleted. The replaced version of the PDSE member is marked for deletion, but it is not until there are no applications accessing that member. For a PDSE which supports generations, open for output, the new member becomes the current generation; a prior generation is deleted deleted if that member exceeds the maximum number of generations.

DISC
Specifies a list of PDSE members which are to be disconnected. Disconnecting members does not modify the data set. The purpose of disconnecting is to release access to a member when access is no longer required. This also frees system resources which are used to track this access. PDSE members are identified by MLT.

IFF
Indicates the “if and only if” function is to be performed. This function replaces a data member of a PDSE if and only if the input timestamp matches the existing member’s creation timestamp. You can use IFF to create a new member (one which does not already exist). In that case, set the compare operand (timestamp) to binary zeros. This function also maintains type descriptor and CCSID attributes. For a PDSE which supports generations, the new member becomes the current generation; a prior generation is deleted if that member exceeds the maximum number of generations.

RG
Indicates that a specified generation for a member of a PDSE is to be replaced with the member being created. Adds the generation if it does not exist. Only the specified generation is replaced, no other generation is deleted.

DG
Indicates that a specified generation for a member of a PDSE is to be deleted; the deleted generation is not retained. When the specified generation is 0, the current member is deleted and the older generations are not affected.

RECOVERG
Specifies that an existing generation is to be recovered so that it becomes the current member.

***list address*—RX-Type Address, (2-12), or (0)**
Specifies the address of the area containing the information that is required by the system to maintain the partitioned data set directory. The size and format of the area depend on the directory action requested as follows::

Add or Replace a Directory Entry: *directory action*=A|R

The *list address* must specify an area at least 12 bytes long and beginning on a halfword boundary. The following illustration shows the format of the area: Table 1

Table 55. List Address Area

NAME	TT	R	C	USER DATA
8 bytes	2 bytes	1 byte	1 byte	0-62 bytes

NAME:
Specifies the member name or alias being added or replaced. STOW is insensitive to the case of the eight characters in the name. All eight character must be used (including padding on the right with blanks if necessary). JCL or other applications might have specific case related requirements. Your program cannot create a name entirely consisting of eight bytes of ‘FF’x.

TT:
Specifies the relative track number where the beginning of the member is located.

R:
Specifies the relative block (record) number on the track that is identified by TT.

Processing Requirements:

- When adding or replacing a member name (alias bit is 0), the system supplies the TTR.

- When adding or replacing an alias name (alias bit is 1), the problem program must supply the TTR.

Note:

- For a PDSE, the TTR field is a token (MLT) that does not represent the physical location of the member in the data set. For a PDSE, the MLT in an alias directory entry must be the MLT of a member already in the directory.
- Alias directory entries in a PDSE must point to the beginning of a member.

C:

Specifies the type of entry (member or alias) for the name, the number of note list fields (TTRNs), and the length in halfwords, of the user data field. The following describes the meaning of the eight bits:

Bit
Meaning

0
Alias bit.

- 0— indicates a member name.
- 1— indicates an alias.

1-2
Indicate the number of TTRN fields (maximum of 3) in your data field.

3-7
Indicates the total number of halfwords in the user data field.

USER DATA FIELD:

The user data field contains the user data for the directory entry. You can use the user data field to provide variable data as input to the STOW macro; there is no specific format for user data, but the Program Management Binder and ISPF do impose a specific format.

Note:

1. User TTRs in the directory entry are not allowed for PDSEs, and will result in an error return code from STOW.
2. The replaced version of a member of a PDSE remains accessible using FIND by TTR or the POINT macro until all connections to it are released. Connections are released when the PDSE is closed or the program that established the connections issues STOW DISC.

Changing the Name of a Member: *directory action*=C

The *list address* must specify the address of a 16-byte area. The first 8 bytes contain the old member name or alias, and the second 8 bytes contain the new member name or alias. Both names must begin in the first byte of their 8-byte area and be padded on the right with blanks, if necessary, to complete the 8-byte field.

Deleting a Directory Entry: *directory action*=D

The *list address* must specify an 8-byte area with the member name or alias to be deleted. The name must begin in the first byte of the area and be padded on the right with blanks, if necessary, to complete the 8 bytes.

When a member of a PDSE is deleted, it remains accessible using FIND by TTR or the POINT macro until all connections to it are released. Connections are released when the PDSE is closed or the program that established the connections issues STOW DISC.

Initializing the Directory: *directory action*=I

Omit the *list address* when the directory action is "I". If the *list address* is specified, it is ignored.

Disconnecting a List of Members: *directory action*=DISC

The list address points to a header and an array of entries to be disconnected. Each entry includes a three byte MLT, a one byte concatenation number, and a one byte status field. The MLT and the concatenation number might have been obtained from a prior BLDL (the fields PDS2TTRP and PDS2CNCT of the directory entry define the respective values).Table 56 on page 337 defines the structure of the list:

Table 56. Disconnect Member List Structure

Offset	Length	Description
X'00'	2	Length of the parameter list (from offset 0). Set by the user.
X'02'	1	Flags X'80' indicates DISC (set by STOW macro).
X'03'	2	Reserved. Must be X'0000' (set by STOW macro).
X'05'	3	DCB address (set by STOW macro).
X'08'	0	Beginning of array of entries to be disconnected. The number of entries is determined from the length of the list.
X'08'	1	Status field (returned from STOW): **X'00'** Member disconnected **X'01'** The supplied MLT is not associated with a connected member. A zero MLT will always return this code. **X'02'** Member represents a partitioned data set **X'03'** Bad concatenation number
X'09'	1	Reserved. Must be X'00'.
X'0A'	3	MLT
X'0D'	1	Concatenation number

Conditionally replacing a member (If and Only If): *directory action*=IFF

The *list address* must be of a 38-byte area. This area includes the address of the DCB, the compare operation (the timestamp), a 16-byte type descriptor, a two byte CCSID, and the address of the new directory entry as described above in Adding or Replacing a Directory Entry: directory action=A | R.

To create a new PDSE data member with the type descriptor and CCSID attributes, a compare operand (timestamp) of binary zeros must be specified on the compare operand. To create a new PDSE data member with new values for the type descriptor and CCSID attributes a valid matching timestamp must be specified on the compare operand to match the timestamp of the last update that is on the disk.

Table 57 on page 337 defines the structure of the list for IFF.

Table 57. Member List Structure for IFF

Offset	Length	Description
X'00'	2	Length of the parameter list (from offset 0). Set by the user.
X'02'	1	Flags - X'40' indicates IFF (set by the STOW macro).
X'03'	2	Reserved. Must be X'0000' (set by the STOW macro).

Table 57. Member List Structure for IFF (continued)

Offset	Length	Description
X'05'	3	DCB address (set by the STOW macro).
X'08'	8	compare operand (timestamp).
X'10'	4	31 bit address of directory entry in PDS2 format. **Note:** Directory entry must indicate primary name.
X'14'	16	Type descriptor. Your application determines the meaning of this field.
X'24'	2	CCSID, coded character set identifier. This field is for application use.

Replacing a Generation: *directory action*=RG

The *list address* must be of a 24-byte area. This area includes the address of the DCB, the member name and generation number, and the address of the new directory entry as described above in Adding or Replacing a Directory Entry: directory action=A | R. Table 58 on page 338 defines the structure of this area.

Table 58. Replace a Generation parameter List Structure

Offset	Length	Description
X'00'	2	Length of parameter list. Set by the user.
X'02'	1	Flags X'20' indicates RG (set by STOW macro).
X'03'	2	Reserved. Must be X'0000' (set by the STOW macro).
X'05'	3	DCB address. (set by the STOW macro).
X'08'	8	Member name, padded on the right with blanks if necessary.
X'10'	4	Absolute generation number of the generation to replace.
X'14'	4	31 bit address of new directory entry. **Note:** The name field of the directory entry is ignored.

Deleting a Generation: *directory action*=DG

The *list address* must be of a 20-byte area. This area includes the address of the DCB, the member name, and generation number. Table 59 on page 338 defines the structure of this area:

Table 59. Delete a Generation parameter List Structure

Offset	Length	Description
X'00'	2	Length of the parameter list. Set by the user.
X'02'	1	Flags X'10' indicates (set by the STOW macro).
X'03'	2	Reserved. Must be X'0000' (set by the STOW macro).
X'05'	3	DCB address (set by the STOW macro).
X'08'	8	Member name, padded on the right with blacks, if necessary.
X'10'	4	Absolute generation number of the generation to delete. Set by the user.

Recovering a Generation: *directory action*=RECOVERG

The *list address* must be of a 20-byte area. This area includes the address of the DCB, the member name and generation number. Table 60 on page 339 defines the structure of this area.

Table 60. Recover a Generation parameter List Structure

Offset	Length	Description
X'00'	2	Length of the parameter list. Set by the user.
X'02'	1	Flags X'08' indicates RECOVERG (set by the STOW macro).
X'03'	2	Reserved. Must be X'0000' (set by the STOW macro).
X'05'	3	DCB address. (set by the STOW macro).
X'08'	8	Member name. Set by the user.
X'10'	4	Generation number of the generation to recover. Set by the user.

Note:

1. For all directory actions other than I (initialize) and DISC (disconnect), you operate on only one member name or alias name at a time with the STOW macro. With the I function, all members of the PDS or PDSE are effectively deleted. For the DISC function, you specify a list of PDSE member identifiers (MLTs) to be disconnected.
2. When adding or replacing a member, if the data set is open for OUTPUT or OUTIN, and the entry to be added is a member name (not an alias), the system writes an end-of-data indication following the member. If the data set is open for update, the entry to be replaced is updated in the directory; no end-of-data record is written. Adding or replacing an alias never writes an end-of-data record.
3. To alter the contents of an existing directory entry, you can issue STOW R if the OPEN option was UPDAT. You must position to the member with the FIND macro before issuing the STOW macro.
4. Whenever you issue the STOW macro, your program must first test all output operations for completion by using the same data control block. If the data set is a PDSE, the CHECK macro can complete before all of the data is on DASD. The STOW macro ensures that all of the member data writes to DASD.
5. For program object PDSEs, you cannot add a new member, replace a member, or replace a generation with STOW. Only the Binder can create new members for program object PDSEs. Recovery of a member generation with STOW is permitted for program object PDSEs. Utilities such as IEBCOPY call the binder when copying programs to create program objects in PDSEs.
6. You can use the STOW macro to disconnect members of a PDSE. Indicate the action with the DISC directory action. If the DISC directory action is specified, the DCB might be open for INPUT, OUTPUT, UPDAT, or OUTIN. Member connections are established by the OPEN, BLDL, FIND, and POINT macros. Member connections are associated with an open DCB. Refer to *z/OS DFSMS Using Data Sets* for information on PDSE connections. All STOW directory actions other than DISC require that the DCB is opened for OUTPUT, UPDAT, or OUTIN.
7. The “generations” directory actions (`RG`, `DG`, and `RECOVERG`) are applicable to PDSE version 2 data sets only. Generations support must be enabled by specifying MAXGENS=n when the PDSE is created.
8. After creating or replacing a member, you can use the GET, GET_ALL, GET_G, or GET_ALL_G functions of DESERV to obtain the member timestamp.
9. Some of the “generations functions” require you to supply an absolute generation number. You can use the GET_G or GET_ALL_G functions of DESERV to obtain the absolute generation numbers. You can also use the FIND macro with the G option to obtain an absolute generation number when you pass a relative generation number.

STOW completion codes

When the system returns control to the problem program, register 15 contains a return code and register 0 contains a reason code in the 2 low-order bytes (Table 61 on page 340

). The high-order bytes of both registers are set to 0. "Directory Action" in the table heading refers to the directory functions add, change, delete, initialize, replace, and disconnect.

Table 61. STOW completion codes other than for IFF

Return Code (15)	Reason Code (0)	Directory Action	Meaning
00 (X'00')	00 (X'00')	A, C, D, R	The update of the directory was completed successfully.
	00 (X'00')	I	The directory was cleared (initialized) successfully.
	00 (X'00')	DISC	Function successful.
04 (X'04')	00 (X'00')	A, C	The directory already contains the specified new name.
	00 (X'00')	DISC	Error detected. Check status fields.
08 (X'08')	01 (X'01')	DISC	Reserved fields not zero.
	02 (X'02')	DISC	Bad length field (either too small for at least one array entry or does not allow for even multiple of array entries).
	03 (X'03')	DISC	Either no function bit is set or reserved function bit is set.
	00 (X'00')	D, R	The specified name is not found.
	C	The specified old name is not found.	
12 (X'0C')	00 (X'00')	A, C, R	No space left in the directory. The entry is not added, replaced, or changed.
12 (X'0C')	01 (X'01')	A, R	For a PDSE, an attempt by STOW to create a member with no records failed because the number of members would have exceeded the maximum allowed.
16 (X'10')	01 (X'01')	A, C, D, I, R	A permanent input or output error was detected. Control is not given to the error analysis (SYNAD) routine.
16 (X'10')	02 (X'02')	A, R	A permanent I/O error occurred while attempting to write the EOF mark after the member. Control is not given to the error analysis (SYNAD) routine.
	04 (X'04')	A, C, D, R	An error occurred while writing data buffered in system buffers. Control is not given to the error analysis (SYNAD) routine.
	1847 (X'737')	A, C, D, I, R	The system found an I/O error while trying to read or write the VTOC."1" on page 342
	2871 (X'B37')	A, C, D, I, R	The system was unable to update the VTOC."1" on page 342
	3383 (X'D37')	A, C, D, I, R	Either no secondary space is available or a DADSM user exit error occurred. The error occurred when trying to write an EOF; all primary space used."1" on page 342

Table 61. STOW completion codes other than for IFF (continued)

Return Code (15)	Reason Code (0)	Directory Action	Meaning
	3639 (X'E37')	A, C, D, I, R	Either no secondary space is available or a DADSM user exit error occurred."1" on page 342
20 (X'14')	00 (X'00')	A, C, D, I, R	The specified data control block is not open or is opened for input, or a DEB error occurred.
24 (X'18')	00 (X'00')	A, C, D, I, R	Insufficient virtual storage was available to perform the STOW function.
28 (X'1C')	00 (X'00')	A, R	The caller attempted to issue add or replace for a member of the Program Management Library, which is a PDSE that contains program objects.
30 (X'1E')	28 (X'1C')	A, R	A stow for an alias is issued for a member that is deleted. However, the member is connected so the delete is pending and the create of the alias failed.
32 (X'20')		A, C, D, I, R	Reason code 8 indicates that an installation exit, IGG_STOW_EARLY, gave the return code the means to fail the STOW invocation. Message IEC997I identifies the exit routine.
34 (X'22')	00 (X'00')	I	In an initialize operation, one or more of the members is placed in a pending delete status.
36 (X'24')	00 (X'00')	A, R	The alias has an invalid TTR (PDSEs only).
40 (X'28')	00 (X'00')	A, R	User-supplied TTRs are in the user data field of the directory entry (PDSEs only).
44 (X'2C')		A, C, D, I, R	Reserved.
48 (X'30')	04 (X'04')	A	The add failed because you cannot add a primary member name while the PDSE is open for update (PDSEs only).
	08 (X'08')	R	The replace failed because you cannot replace a primary member name while the PDSE is open for update and the specified name does not exist (PDSEs only).
	12 (X'0C')	R	The replace failed because you cannot replace an alias name if it is the same name as the primary member (PDSEs only).
	16 (X'10')	A, R	The add or replace failed when attempting to add or replace an alias, but the member identified by the TTR did not exist (PDSEs only).
	20 (X'14')	R	The replace failed when attempting to replace a primary member name while the PDSE is open for update and the member name identified an existing alias (PDSEs only).
	24 (X'18')	R	The replace failed when attempting to replace a primary member name while the PDSE is open for update, but the input TTR has not been defined for that member (PDSEs only).

Table 61. STOW completion codes other than for IFF (continued)

Return Code (15)	Reason Code (0)	Directory Action	Meaning
52 (X'34')	00 (X'00')	I	One or more members were placed in a pending delete state; the space taken by those modules is not immediately available for reuse.

Note:

1. See *z/OS MVS System Codes* for more information on abend codes X'737', X'B37', X'D37', and X'E37'.

The following return and reason codes might be returned from STOW IFF:

Table 62. STOW IFF completion codes

Return Code (15)	Reason Code (0)	Description
00 (X'00')	00 (X'00')	STOW IFF successful. The compare operands matched and the directory was updated successfully, the member was createdor updated.
04 (X'04')	00 (X'00')	STOW IFF successful. The input compare operand was zero, the input member name did not exist. The directory was updated successfully, the member was added.
08 (X'08')	00 (X'00')	STOW IFF failed. Either: 1. The input compare operand was zero, but the input member name exists. 2. The input compare operand was not zero, the input member name exists, but the compare operand (Timestamp) does not match that of the existing member.
28 (X'1C')	04 (X'04')	The DCB indicates a PDS. STOW IFF is not supported for PDS members.

The following return and reason codes might be returned from STOW RG, DG, and RECOVERG:

Table 63. STOW RG, DG and RECOVERG Completion Codes

Return Code (15)	Reason Code (0)	Directory Action	Description
00 (X'00')	00 (X'00')	RG, DG, RECOVERG	Function successful.
08 (X'08')	00 (X'00')	DG	Generation does not exist, not deleted.
08 (X'08')	00 (X'00')	RG	Name exists but generation did not exist. Add performed.
08 (X'08')	00 (X'00')	RECOVERG	Generation does not exist, no recovery performed.
48 (X'30')	28 (X'1C')	RG	Data set is not defined to have generations.

Table 63. STOW RG, DG and RECOVERG Completion Codes (continued)

Return Code (15)	Reason Code (0)	Directory Action	Description
48 (X'30')	32 (X'20')	RG	No generation has ever existed. Generations must exist before an RG operation can be performed.
48 (X'30')	36 (X'24')	RG	The generation specified is out of range. It is older than the oldest generation and generation limit has been reached.
48 (X'30')	40 (X'28')	RG	The generation specified is out of range. It is newer than the newest generation.

SYNADAF—Perform SYNAD analysis function (BDAM, BISAM, BPAM, BSAM, EXCP, QISAM, and QSAM)

The SYNADAF macro is used in an error analysis routine to analyze permanent input/output errors.The routine can be a SYNAD exit routine specified in a data control block for BDAM, BISAM, BPAM, BSAM, QISAM, QSAM, or a routine specified in a DCBE for BPAM, BSAM, QSAM, or a routine that is entered directly from a program that uses the EXCP or XDAP macro. (The EXCP and XDAP macros are described in *z/OS DFSMSdfp Advanced Services*.)

The SYNADAF macro uses register 1 to return the address of an area containing a message. The message describes the error, and can be printed by a later PUT, WRITE, or WTO macro. The message consists mainly of EBCDIC information and is in variable-length record format. The format of the area is shown following the descriptions of the SYNADAF parameters.

For extended format data sets, PDSEs, or UNIX files, SYNADAF returns an additional message. The first message contains an 'S' at offset 127 to indicate that the second message exists. The second message is located at 8 bytes past the end of the first message. This second message provides additional information to further describe the error. It can be printed with another PUT, WRITE, or WTO macro.

The system does not save registers in the save area whose address is in register 13. Instead, it provides a save area for its own use, and then makes this area available to the error analysis routine. The system returns the address of the new save area in register 13 and in the appropriate location (third word) of the previous save area. The system also stores the address of the previous save area in the appropriate location (second word) of the new save area.

When the SYNADAF macro is issued in 31-bit addressing mode, the caller must ensure that the input save area address in register 13 is a valid 31-bit address. This would be true unless your program changes it.

The SYNADAF macro passes parameters to the system in registers 0 and 1. When used in a SYNAD exit routine, you should code the SYNADAF macro at the beginning of the routine. (See *z/OS DFSMS Using Data Sets* for information on the SYNAD exit routine.) For BISAM and QISAM, the SYNAD exit routine has to set up these parameters as explained under PARM1 and PARM2. To save these parameters for use by the SYNAD exit routine, the system stores them in a parameter save area that follows the message buffer as shown in the message buffer format. The second message immediately follows these two parameters.

The system does not alter the return address in register 14. On return from SYNADAF, the high order byte of register 15 has been modified. The low order three bytes are unchanged.

Restriction: Callers of SYNADAF in 31-bit addressing mode must either not use register 15 as a base register or restore the high order byte of register 15 on return from SYNADAF.

When a SYNADAF macro is used, you must use a SYNADRLS macro to release the message area and save area, and to restore the original contents of register 13.

The format of the SYNADAF macro is:

[*label*]	SYNADAF	ACSMETH={BDAM [,PARM1=*parm register*] [,PARM2=*parm register*]} {BPAM [,PARM1=*parm register*] [,PARM2=*parm register*]} {BSAM [,PARM1=*parm register*] [,PARM2=*parm register*]} {QSAM [,PARM1=*parm register*] [,PARM2=*parm register*]} {BISAM [,PARM1=*dcbaddr*] [,PARM2=*decbaddr*]} {EXCP [,PARM1=*iobaddr*] [,PARM2=*iobeaddr*]} {QISAM [,PARM1=*dcbaddr*] [,PARM2=*parm register*]}

ACSMETH=BDAM, BPAM, BSAM, QSAM, BISAM, EXCP, or QISAM
specifies the access method that is used to perform the input/output operation when the SYNADAF macro performs error analysis. Code ACSMETH=EXCP if your program used an EXCP or XDAP macro.

Recommendation: Do not use BISAM, EXCP, XDAP or QISAM.

PARM1=*parm register*, *iobaddr*, or *dcbaddr*—(2-12) or (1)
specifies the address of information that is dependent on the access method being used. For BDAM, BPAM, BSAM, or QSAM, the parameter specifies a register containing the information that was in register 1 on entry to the SYNAD routine. For BISAM or QISAM, it specifies the address of the data control block. For EXCP, it specifies the address of the input/output block. If the parameter is omitted, PARM1=(1) is assumed.

PARM2=*parm register, dcbaddr, iobaddr*, or *iobeaddr*—(2-12), (0), or RX-Type
specifies the address of additional information that is dependent on the access method being used. For BDAM, BPAM, BSAM, QISAM, and QSAM, the parameter specifies a register containing the information that was in register 0 on entry to the SYNAD exit routine. For BISAM, the parameter specifies a register containing the information that was in register 1 on entry to the SYNAD exit routine (the address of the DECB). For EXCP, the parameter specifies the address of the IOBE only if your program provided the IOBE with the EXCP, EXCPVR or XDAP macro. If your program used 31-bit format 1 CCWS with an IOBE, this parameter is necessary. For all access methods if the parameter is omitted, PARM2=(0) is assumed.

To correctly load the registers for SYNADAF for BISAM, code these two instructions before issuing the SYNADAF macro:

```
    LR   0,1         GET DECB ADDRESS
    L    1,8(1)      GET DCB ADDRESS
```

SYNADAF completion codes

When the system returns control to the problem program, the low-order byte of register 0 contains a completion code. The 3 high-order bytes of register 0 are set to 0.

The SYNADAF completion codes are:

Completion Code (0)	Meaning
00 (X'00')	Successful completion. Bytes 8 through 23 of the message area contain blanks or the first part of a VSAM physical error message for ISAM.
04 (X'04')	Successful completion. If you are not using the large block interface (LBI), bytes 8 through 13 of the message area contain binary data. If you are using the large block interface (LBI), bytes 8 through 23 of the message area contain binary data.
08 (X'08')	Unsuccessful completion. The message can be printed, but some information is missing in bytes 50 through 127 and is represented by asterisks. Bytes 8 through 23 will be blanks (X'40') if no data was read. If data was read without LBI, then bytes 8 through 11 contain the binary address of the data, bytes 12 and 13 contain the binary length of the data read, and bytes 14 through 23 contain blanks. If data was read with LBI, then bytes 8 through 23 contain two eight-byte binary fields. The low order 31 bits of the first field contain the address of the data and the high order 33 bits contain binary zeroes. The low order 32 bits of the second field contain the length of the data read and the high order 32 bits contain binary zeroes.

Message buffer format

Figure 7 on page 346 shows the format of the message area.

You can use the IGGSYNT mapping macro for the area that SYNADAF returns. The DSECT name is SYNT. The symbols are shown in Figure 7 on page 346.

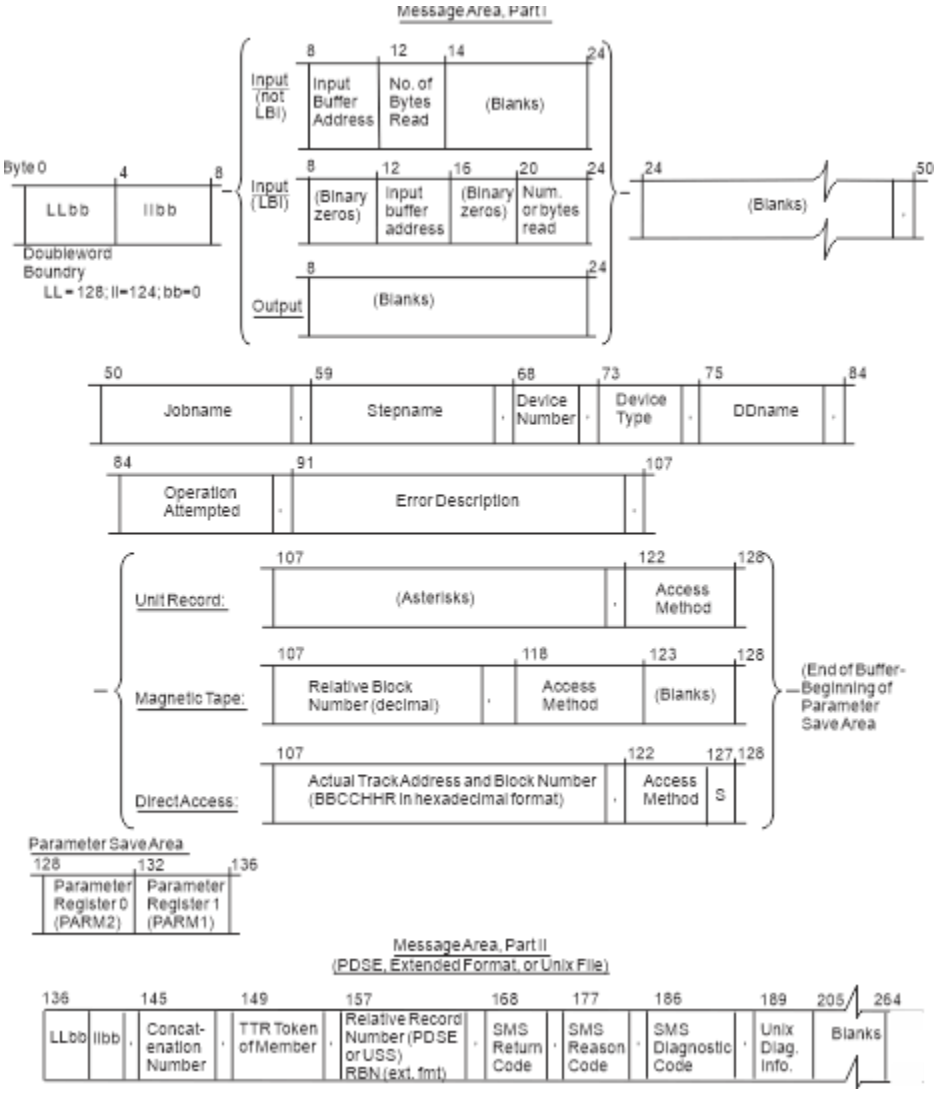


Figure 7. Message Buffer Format

The address of the message area is returned in register 1. The area comes in two parts. Each part is a separate variable length record. If the data set being analyzed is not a PDSE, an extended format data set, or a UNIX file, only the first area is created. Otherwise, both areas are created. The first area is 120 characters long, and begins with a field of blanks; you can use the blank field to add your own remarks to the message. The text of the second area begins 8 bytes past the end of the first message. It is 128 characters long and ends with several blanks (reserved for later definition by IBM).

Fields contain EBCDIC characters unless otherwise stated here. For most character fields if the system cannot get the information, the field contains asterisks.

If you suspect a system software error, report the SMS return code, reason code and diagnostic code and UNIX diagnostic information to your IBM service representative.

Table 64. Message Area Details

Offset	Length		Description
0	2	SYNBDW1	Length of variable-length block in binary. Always 128.
2	2		X'0000'.
4	2	SYNRDW1	Length of variable-length record in binary. Always 124.
6	2		X'0000'.

Table 64. Message Area Details (continued)

Offset	Length		Description
Input, not using LBI and access method is not BISAM or QISAM			
8	4		Data address (binary). Might be zero.
8	120	SYNMSG	Total text.
8	16	SYN1READ	Reading information with LBI.
8	16	SYN2READ	Reading information with LBI.
8	6	SYNREAD	Reading information without LBI.
8	4	SYNRDERR	Buffer address if reading without LBI.
12 (C)	2		Number of bytes read (binary). Might be zero.
12	4	SYN2RDER	Buffer address if reading with LBI.
12	2	SYNBYTRD	Number of bytes read without LBI.
14 (E)	35		Blanks.
14	10	SYNBLNKI	Blanks.
Input, using LBI			
8	4		Binary zeroes.
12 (C)	4		Data address (binary).
16 (10)	4		Binary zeroes.
20 (14)	4		Number of bytes read (binary).
20	4	SYN2BYTR	Number of bytes read with LBI.
20	4	SYN2BYTR	Number of bytes read with LBI.
24 (18)	25		Blanks.
24		SYNWAREA	Blanks.
24	25	SYNWAREA	Blanks.
Output or access method is BISAM or QISAM			
8 (8)	41		Blanks. If BISAM or QISAM, this might contain a VSAM message as described in "Reason code (physical errors)" on page 138.
All			
49 (31)	1	SYNCMMA1	Comma.
50 (32)	8	SYNJOBNM	Job name.
58 (3A)	1	SYNCMMA2	Comma.
59 (3B)	8	SYNSTPNM	Step name.
67 (43)	1	SYNCMMA3	Comma.
68 (44)	4	SYNUNTID	Device number in hexadecimal or one of the following values: JES (spooled data set or a subsystem data set (SUBSYS=)), OMVS (UNIX file or directory) or N/A (BISAM or QISAM).
72 (48)	1	SYNCMMA4	Comma.
73 (49)	1	SYNDVTYP	Device class: D (direct access), T (tape), U (unit record) or * (other). For BISAM or QISAM, the previous byte and this byte contain this value: DA.
74 (4A)	1	SYNCMMA5	Comma.

Table 64. Message Area Details (continued)

Offset	Length		Description
75 (4B)	8	SYNDDNM	DD name.
83 (53)	1	SYNCMMA6	Comma.
84 (54)	6	SYNOPRTN	Operation attempted such as READ, WRITE, GET, PUT, POINT or ENDREQ.
90 (5A)	1	SYNCMMA7	Comma.
91 (5B)	15	SYNERROR	Error description. Might be N/A or NOT APPLICABLE. See "SYNADAF error descriptions" on page 351. If it indicates a padding error, then the data set is extended format and the data was damaged when it was written. This might be due to hardware error, an operator cancel or time out while the control unit was transferring data.
106 (6A)	1	SYNCMMA8	Comma.
Direct access			
107 (6B)	14	SYNPOS	Actual track address and block number (BBCCHHR). If compressed format data set, this is a relative block number (RBN) in bytes 107-113. If a logical error in a compressed format data set and not a physical error, this may contain asterisks. If the error was in POINT or backspace, this can be TTR UNKNOWN. If BDAM and the error was an invalid request, character zeroes. If UNIX file, low order portion of file offset in hex.
121 (79)	1	SYNCMMA9	Comma.
122 (7A)	5	SYNACCSS	Access method.
127 (7F)	1	SYNSMS	Message code. If S, then a second message exists beginning at offset 136.
Magnetic tape			
107 (6B)	10	SYNPOSM2	Relative block number in data set (decimal).
117 (75)	1	(No name)	Comma.
118 (76)	5	(No name)	Access method.
123 (7B)	5	(No name)	Blanks.
127	1	SYNBLNK2	Blank.
Not direct access or magnetic tape			
107 (6B)	14	SYNPOS	Asterisks.
121 (79)	1	SYNCMMA9	Comma.
122 (7A)	6		Access method.
122	5	SYNACCSS	Access method type.
127	1	SYNBLNK2	Blank.
128	4	SYNPRMR1	
132	4	SYNPRMR2	
136	0	SYNEND1	
138	2		
140	2	SYNRDW2	

Table 64. Message Area Details (continued)

Offset	Length		Description
142	2		
144	0	SYNMSG2	Second text.
144	1	SYN2CMA1	
145	3	SYNCNCAT	
148	1	SYN2CMA2	
149	7	SYNMLT	
156	1	SYN2CMA3	
157	10	SYNRRN	
167	1	SYN2CMA4	
168	8	SYNSMSRC	
176	1	SYN2CMA5	
177	8	SYNSMSRS	
185	1	SYN2CMA6	
186	2	SYNAMAF	
188	1	SYN2CMA7	
189	16	SYN2DIAG	
205	59	SYNBLNKS	
264	0	SYNEND2	End of second text.
All device classes			
128 (80)	4		Parameter register 0 (PARM2) passed to SYNADAF. Binary.
132 (84)	4		Parameter register 1 (PARM1) passed to SYNADAF. Binary.
The rest of the area exists only if the byte at +127 contains S. Currently this occurs only for PDSE, extended format data set or z/OS UNIX file.			
136 (88)	2		Length of variable-length block in binary. Always 128.
138 (8A)	2		X'0000'.
140 (8C)	2		Length of variable-length record in binary. Always 124.
142 (8E)	2		X'0000'.
144 (90)	1		Comma.
145 (91)	3		Concatenation number (first data set is 0).Asterisks if a z/OS UNIX file.
148 (94)	1		Comma.
149 (95)	7		Member locater token (MLT, simulated TTR) in hex if PDSE or UNIX directory with BPAM. First byte is blank. Zero if extended format data set.
156 (9C)	1		Comma.
157 (9D)	10		A decimal number. Record number in PDSE member, record number in UNIX file or block number in extended format or compressed format data set. Leading zeroes are blanks. The first record or block is 1.

Table 64. Message Area Details (continued)

Offset	Length	Description
167 (A7)	1	Comma.
168 (A8)	8	One of the following hex codes: • SMS return code for a PDSE. Refer to *z/OS DFSMSdfp Diagnosis*. • Media manager return code for an extended format data set. Refer to *z/OS DFSMSdfp Diagnosis*. • Feedback code for a UNIX file. The feedback code is similar to what is described for RPLFDBWD for V SAM in "Record management return and reason codes" on page 120. The third and fourth digits (return code) are one of these values like RPLRTNCD: **08** Logical error **0C** Physical error. For a UNIX file the seventh and eighth digits are described in Table 65 on page 350.
176 (B0)	1	Comma.
177 (B1)	8	SMS reason code in hex unless a UNIX file with a physical I/O error. In that case, the name of the failing UNIX service.
185 (B9)	1	Comma.
186 (BA)	2	SMS diagnostic code. Contains "**" if not available. Otherwise refer to Table 67 on page 351.
188 (BC)	1	Comma if UNIX file. Blank otherwise.
189 (BD)	16	UNIX diagnostic information if UNIX file and a physical I/O error, otherwise blanks. Format is xxxx-yyyyyyyy, which are the z/OS UNIX return and reason codes in hex.
205 (CD)	59	Blanks.

The two digits in the feedback code at offset 174 and 175 for a UNIX file are hexadecimal values like RPLERRCD. Refer to Table 65 on page 350 and Table 66 on page 351.

Table 65. Feedback Code at Offset 174 and 175 for a Logical Error

Offset	Description
04 (4)	End of data.
10 (16)	Beyond end of file or invalid record.
1C (28)	System is not able to extend the file.
20 (32)	Invalid position in file.
28 (40)	Insufficient virtual storage.
2C (44)	Buffer too small.
30 (48)	Invalid option in internal system control block.
44 (68)	Invalid internal system control block.
48 (72)	Invalid option in internal system control block.

Table 65. Feedback Code at Offset 174 and 175 for a Logical Error (continued)

Offset	Description
54 (84)	Invalid option in internal system control block.
68 (104)	Invalid option in internal system control block.
DA (218)	Unrecognized internal system return code.

Table 66. Feedback Code at Offset 174 and 175 for a Physical Error

Offset	Description
04 (4)	Read error.
10 (16)	Write error.
DA (218)	Unrecognized internal system return code or option.

Table 67. Hexadecimal Codes in the SMS Diagnostics Code Field at Offset 186

Code (hex)	Description
01	System logic error in asynchronous routine.
02	System logic error (program check).
03	System logic error. Bad SMS return code.
04	Invalid locater token passed to POINT. User error.
05	POINT issued when no member was connected. User error.
06	The DCB was open for output and POINT was called with an MLT for other than the current member. User error.
07	POINT issued when I/O was outstanding. User error.
08	POINT issued with an RLT that is too big for the file or member. User error.
09	Connect or reconnect unable to get file lock for POINT. Might be a user error.
0A	POINT issued with invalid MLT. User error.
0B	Padding error for extended format data set.
0C	I/O error for extended format data set.
0D	I/O error for a compressed format data set.
0E	Data set is compressed format and READ or WRITE request was for an RBN that exceeds the X'FFFFFF' limit and BLOCKTOKENSIZE=LARGE was not coded.

SYNADAF error descriptions

Table 68 on page 351 describes some of the internal system logic. Future technical updates may cause changes to sample phrases that describe the existing or new error conditions. These sample phrases are at offset 91. IBM reserves the right to change the internal system logic and these phrases at any time.

Table 68. SYNADAF – Sample Phrases

Error Message	Description
NO ERROR STATUS	There was no error.
OUT OF EXTENT	Trying to read or write outside the allocated space.

Table 68. SYNADAF – Sample Phrases (continued)

Error Message	Description
PREVIOUS D.E. ERROR	After the previous channel program completed successfully with channel end status, the I/O subsystem reported a device end status with an error.
PURGED REQUEST	The I/O request terminated early by software.
CHANNEL CTL CK	Channel-Control Check. Machine malfunction affecting channel-subsystem controls.
INTRFACE CTL CK	Interface-Control Check. Invalid signal has occurred on the channel path. Usually indicates a malfunctioning of an I/O device.
PROGRAM CHECK	Program Check. Programming error in channel program such as: invalid command code, invalid address or invalid flags.
PROT CHECK	Protection Check. The channel program attempted a storage access that is prohibited by the protection mechanism. This applies to fetching of CCWs, IDAWs, and output data, and to the storing of output data.
CHAINING CHECK	Chaining Check. I/O data rate is too high for the resolution of data addresses.
END OF FILE	Unit Exception. Tape mark or disk file mark was read. Sometimes the system simulates this due to end of DASD extent or end of tape.
WRNG. LEN. RECORD	Wrong Length Record. Length of block differs from what the access method expected.
PADDING ERROR	During input the access method determined that the block has an invalid content. This implies an error when the block was being written. Perhaps there was a power or control unit failure during the write.
EQUIPMENT CHECK	Equipment check. An equipment malfunction was detected between the I/O interface and the I/O medium.
BUS OUT CHECK	Bus out check. The I/O device or the control unit recognized certain error conditions on the channel path.
COMMAND REJECT	Command reject. The I/O device has detected a channel programming error.
INTERV REQUIRED	Intervention required. The command could not be executed because of a condition that requires human intervention at the I/O device.
DATA CHECK	Data check. The control unit or I/O device detected invalid data. Incorrect data might have been placed in virtual storage or recorded at the device.
OVERRUN	Overrun. For a request for service from the I/O device the channel subsystem failed to respond to the control unit in the anticipated time interval.
SEEK CHECK	Seek check or incomplete domain. The expected number of data transfer commands specified by the locate record count parameter were not received.
TRACK OVERRUN	Imprecise ending. In the domain of a locate record or locate record extended command, the exception status is for a previously completed CCW.
IMPRECISE END	Imprecise ending. In the domain of a locate record or locate record extended command, the exception status is for a previously completed CCW.
CYLINDER END	Cylinder end. A command tried to continue past the end of the cylinder.
INVALID SEQ	Record sequence error. Invalid sequence with seven track tape or the block ID shows that the block was read out of sequence.
NO RECORD FOUND	No record found. Requested block was not found.

Table 68. SYNADAF – Sample Phrases (continued)

Error Message	Description
FILE PROTECT	File protect. Disk: An operation violated the define extent command or the file mask in a locate record command. Typically this means the channel program tried to operate outside the data set. Tape: The tape cartridge is protected against writing.
MISSING A.M.	Missing address marker (if reading) or write inhibited (if writing).
OVERFLOW INCOMP	Invalid track format, trying to write too much data on a track or the format of the data on the track was not valid for the type of operation being performed.
WORD COUNT ZERO	Word count zero on reel tape or deferred unit check on cartridge tape. Failure is associated with a unit check status that is not related to the execution of the current command.
DEFERRED U.C.	Deferred unit check. Error in a previous command.
DATA C.CHECK	Data converter check on seven-track tape or 'assigned elsewhere' on cartridge tape. 'Assigned elsewhere' means the device is associated with a unit check status. It is disabled by dynamic partitioning on the selected channel path.
ASSIGNED OTHER	Assigned elsewhere. The device is disabled by dynamic partitioning on the selected channel path.
NOT CAPABLE	Not capable. The tape drive cannot read a data block on a tape. Either the recording-format identification at the beginning of tape is missing or not readable, or the data block exceeds the maximum block size.
NOT READY	Magnetic tape not ready.
7-TRACK TAPE	Record Sequence Error. The block ID shows that the block being read is out of sequence.
AT LOAD POINT	At beginning of tape.
WRITE STATUS	Write Mode. The most recent command was a write-type command.
READY	Drive is online.
NOT IN SRCH.LMT	The record was not found.
SPACE NOT FOUND	Either there is no dummy record when adding a format-F record or there is no space available when adding a format-V or format-U record.
I/O ERROR	I/O error.
END OF DATA	The record requested is an end of data record.
ERROR NOT I/O	Error other than for data transfer. The access method is BDAM, which set exception code bit 6. One possibility is that there was a unit check and not data check and the condition was not "no record found". Another possibility is that the failure was not a unit check and not a wrong length record.
EXCLUSIVE CNTRL	A WRITE, type DIX or DKX, has occurred for which there is no previous corresponding READ with exclusive control.
INPUT DCB	A WRITE was attempted for an input data set.
LIMCT=0-EX.SRCH	An extended search was requested, but LIMCT was zero.
CAPACITY RECORD	Writing a capacity record (R0) was attempted.

Table 68. SYNADAF – Sample Phrases (continued)

Error Message	Description
INCORRECT KEY	A READ or WRITE with key was attempted, but either KEYLEN was zero or the key address was not supplied.
INVALID OPTIONS	The READ or WRITE macro request options conflict with the OPTCD or MACRF parameters.
FIX.LEN.KEY'FF'	A WRITE (add) with fixed length was attempted with the key beginning with X'FF'.
UNKNOWN ERROR	Indeterminate error.

SYNADRLS—Release SYNADAF buffer and save areas (BDAM, BISAM, BPAM, BSAM, EXCP, QISAM, and QSAM)

The SYNADRLS macro releases the message buffer, parameter save area, and register save area provided by a SYNADAF macro. It must be used to perform this function whenever a SYNADAF macro is used.

When the SYNADRLS macro is issued, register 13 must contain the address of the register save area provided by the SYNADAF macro. The control program loads register 13 with the address of the previous save area, and sets word 3 of that save area to 0. Thus, when control is returned, the save area pointers are the same as before the SYNADAF macro was issued.

The SYNADRLS macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses. When the SYNADRLS macro is issued in 31-bit addressing mode, the caller must ensure that the input save area address in register 13 is a valid 31-bit address. This would be true unless your program changes it.

On return from SYNADRLS, register 15 will be unpredictable. Therefore, callers in 31-bit addressing mode must either not use register 15 as a base register or must restore register 15 on return from SYNADAF or SYNADRLS.

The format of the SYNADRLS macro is:

[*label*]	SYNADRLS	b

SYNADRLS completion codes

When the system returns control to the problem program, the low-order byte of register 0 contains a completion code. The 3 high-order bytes of register 0 are set to 0.

The SYNADRLS completion codes are:

Completion Code (0)	Meaning
00 (X'00')	Successful completion.
08 (X'08')	Unsuccessful completion. The buffer and save areas were not released; the contents of register 13 remain unchanged. Register 13 does not point to the save area provided by the SYNADAF macro, or this save area is not properly chained to the previous save area.

SYNCDEV—Synchronize device (BSAM, BPAM, QSAM, EXCP)

Tape data sets

The SYNCDEV macro allows you to suspend your program until the control unit cache contents have been written to the magnetic tape cartridge. (All cartridge tapes support buffered write mode.) This synchronizes your program's data and the data on the tape. When you synchronize your data, you ensure that the system checks your data and does not lose any of it when the system writes the data out to storage. Thus, you can avoid several problems you might have if your data is not synchronized.

For example, if your data is not synchronized, your program could update other data sets before the records that were sent to the buffer have finished being written onto tape. How much data is left in the buffer depends on how fast the tape moves. If your program and the tape drive fail, then the tape and the other data set would have inconsistent contents. The problems discussed in this paragraph rarely occur. The system automatically synchronizes the data when going to a new volume or when the data set is closed. The use of SYNCDEV can severely degrade performance of the tape drive.

You can use the SYNCDEV macro to:

- Request information regarding synchronization.
- Demand synchronization if the specified number of data blocks are buffered. If more blocks are buffered than were specified, the system stays in control until all the blocks are written to the tape or it detects an I/O error. If the same amount or fewer blocks are buffered, buffering is not affected. With BSAM your program should issue CHECK or WAIT macros for all outstanding writes. With EXCP, your program should wait for completion of all writes or purge them. SYNCDEV purges outstanding I/O.

Note: Demands for synchronization are ignored if the drive is in read mode.

DASD data sets

The SYNCDEV macro allows you to synchronize data from the following types of DASD data sets when open for update or output:

- PDSEs
- Compressed format data sets
- UNIX files

All other types of data sets are not supported.

For DASD data sets, requests for synchronization information or for partial synchronization cause complete synchronization. The keywords ABUFBLK, BUFBLK, and INQ are ignored. Use the SYNCDEV macro if you need to ensure that a specific record is on DASD at a specific time.

Data is always synchronized at CLOSE (or STOW for PDSEs opened with DSORG=PO).

SYNCDEV guarantees that the data from previously checked output requests has been written to DASD. If you are using BSAM, you still need to issue a CHECK for each WRITE before issuing the SYNCDEV macro. When using SYNCDEV with QSAM, any records left in your current buffer are held if that buffer is only partially filled.

Instead of using the SYNCDEV macro, you can specify "Guaranteed Synchronous Write" through storage class to synchronize the data if the PDSE member is open for update or if the data set is a compressed format data set open for output. See *z/OS DFSMSdfp Storage Administration* for more information.

Restriction: Using SYNCDEV or of "Guaranteed Synchronous Write" can severely degrade performance of data transfer.

The SYNCDEV macro can be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The format of the SYNCDEV macro is:

[*label*]	SYNCDEV	DCB=*addr* [,{ABUFBLK=*addr*| BUFBLK={*maximum buffer depth*| *0*}}] [,INQ={YES|<u>NO</u>}]

DCB=*addr*—A-Type address or (2-12)
specifies the address of the data control block. When SYNCDEV is issued in 31-bit addressing mode, the input DCB address must be a clean 31-bit address.

ABUFBLK=*addr*| BUFBLK={*maximum buffer depth*|*0*}
specifies the maximum number of data blocks that can remain buffered.

ABUFBLK=*addr*—A-Type address or (2-12)
specifies the address of a halfword on a halfword boundary containing a value that specifies the maximum number of data blocks that are buffered. This parameter has no effect on DASD.

This inquiry call also synchronizes DASD data sets as if BUFBLK=0 were coded. When issued in 31-bit addressing mode, the input ABUFBLK address must be a clean 31-bit address.

BUFBLK={*maximum buffer depth*|*0*}
specifies the maximum number of data blocks that are buffered. This number can be an absolute value from 0 to 65535. The BUFBLK value can be in the 2 low-order bytes of a register (2-12). This parameter has no effect on DASD.

0
If neither ABUFBLK nor BUFBLK is specified, the number of data blocks that are be buffered defaults to 0, and no data blocks are buffered.

INQ={YES|<u>NO</u>}
specifies whether this is a request for information about the degree of synchronization or a request for synchronization. This parameter has no effect on DASD.

YES
specifies an inquiry about how many data blocks are in the buffer. This inquiry call also synchronizes DASD data sets and sets the buffer depth to 0.

<u>NO</u>
specifies a request for synchronization based on the number of data blocks that can be buffered as specified in ABUFBLK or BUFBLK.

Register 0 returns the number of blocks that were in the buffer when SYNCDEV began.

SYNCDEV—List form

The list form of the SYNCDEV macro is:

[*label*]	SYNCDEV	[DCB=*addr*] [,BUFBLK={*maximum buffer depth*|*0*}] [,INQ={YES|<u>NO</u>}] ,MF=L

DCB=*addr*—A-Type address

BUFBLK={*maximum buffer depth*|*0*}

INQ={YES|<u>NO</u>}

MF=L
generates a parameter list containing no executable instructions. The list can be used as input and can be modified by the execute form of the SYNCDEV macro.

SYNCDEV—Execute form

The execute form of the SYNCDEV macro is:

[*label*]	SYNCDEV	[DCB=*addr*] [,{ABUFBLK=*addr*| BUFBLK={*maximum buffer depth*|*0*}}] [,INQ={YES|NO}] ,MF=(E,*addr*)

DCB=*addr*—RX-Type address or (2-12)

ABUFBLK=*addr*| BUFBLK={*maximum buffer depth*|*0*}

INQ={YES|NO}

MF=(E,*addr*)
specifies the execute form of SYNCDEV.

***addr*—RX-Type address, or (2-12)**
specifies the address for the parameter list.

SYNCDEV completion codes

When the system returns control to your problem program, the low-order byte of register 15 contains a return code. If register 15 is nonzero, the low-order byte of register 0 contains a reason code.

The SYNCDEV return and reason codes are:

Return Code (15)	Reason Code (0)	Meaning
00 (X'00')		Successful completion. Register 0 always contains 0.
04 (X'04')	01 (X'01')	Incorrect parameter.
04 (X'04')	02 (X'02')	Incorrect DCB or a DEB error.
	03 (X'03')	System error occurred.
	04 (X'04')	Possible system error.
	05 (X'05')	1) Device does not support buffering, *or* 2) SYNCDEV was issued for a DASD data set that is not supported.
	06 (X'06')	Device does not support block IDs for tape data.
	07 (X'07')	Invalid environment was detected by an SMS service while processing a DASD data set. Probable system error.
	08 (X'08')	This is an informational message that is issued when using QSAM to process a DASD data set. SYNCDEV completed successfully. Logical records left in your QSAM buffer might not have been written to DASD.
	11 (X'0B')	Unsuccessful call to ESTAE macro.
	12 (X'0C')	Insufficient virtual storage available.
08 (X'08')	00 (X'00')	Permanent I/O error during read block ID or synchronize command.
12 (X'0C')	00 (X'00')	Permanent I/O error on the last channel program with loss of data (for tape data only). **Note:** If you specified a SYNAD option in the DCB or DCBE and issue a PUT or CHECK macro after this error occurs, the system does not call the SYNAD routine unless an I/O error recurs.

Return Code (15)	Reason Code (0)	Meaning
	01 (X'01')	An I/O error was detected by a previous output request while processing a DASD data set.

TRUNC—Truncate buffer (QSAM output—fixed or variable-length blocked records and BSAM)

For QSAM: The TRUNC macro causes the current output buffer to be regarded as full. The next PUT or PUTX macro specifying the same data control block uses the next buffer to hold the logical record.

A TRUNC macro issued against a PDSE does not create a short block because the block boundaries are not saved on output. On input, the system uses the block size specified in DCBBLKSI for reading the PDSE.

When a variable-length spanned record is truncated and logical record interface, or extended logical record interface, is specified (that is, if BFTEK=A is specified in the DCB macro, or if a BUILDRCD macro is issued, or if DCBLRECL=0K or nnnnnK is specified), the system segments and writes the record before truncating the buffer. Therefore, the block being truncated is the one containing the last segment of the spanned record.

The TRUNC macro is ignored if it is used for unblocked records, if it is used when a buffer is full, if it is used without an intervening PUT or PUTX macro, or when it is used with UNIX files.

For BSAM on DASD: The TRUNC macro causes any queued READs or WRITEs to be issued although the accumulation limit has not been reached. See the DCBE MULTACC parameter.

The BSAM issuer of TRUNC should ensure that if the DCB address is supplied in a register, it must be a valid 31-bit address even if the issuer is not in 31-bit mode. If the buffers are above the line, TRUNC must be issued in 31-bit mode.

If you issue a TRUNC macro for a DCB for a spooled, tape, dummy, or compressed format data set, it has no effect.

Recommendation: If a WAIT is issued while DCBE MULTACC is specified, it should be preceded by a TRUNC macro because future levels of the system might require it.

Do not issue a TRUNC macro when using BSAM to create a direct (BSAM) data set or with BFTEK=R (when reading a direct data set).

The TRUNC macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The format of the TRUNC macro is:

[*label*]	TRUNC	*dcb address*

***dcb address*—RX-Type Address, (2-12), or (1)**
specifies the address of the data control block for the sequential data set opened for QSAM output or for BSAM. For QSAM, the record format in the data control block must not indicate standard blocked records (RECFM=FBS). When issued in 31-bit addressing mode, the input DCB address must be a clean 31-bit address.

WAIT—Wait for one or more events (BDAM, BISAM, BPAM, and BSAM)

The WAIT macro informs the control program that performance of the active task cannot continue until one or more specific events, each represented by a different ECB (event control block), have occurred. The ECBs represent completion of I/O processing associated with a READ or WRITE macro. ECBs are located at the beginning of access method DECBs (data event control blocks), so that the DECB name provided in READ and WRITE macros is also used for WAIT. (A description of the ECB is found in "Status

information following an input/output operation” on page 373.For information on when to use the WAIT macro, see *z/OS DFSMS Using Data Sets*.)

The control program takes the following action:

- For each event that has already occurred (each ECB is already posted), the count of the *number of events* is decreased by 1.
- If *number of events* is 0 when the last event control block is checked, control is returned to the instruction following the WAIT macro.
- If *number of events* is not 0 when the last ECB is checked, control is not returned to the issuing program until sufficient ECBs are posted to bring the number to 0. Control is then returned to the instruction following the WAIT macro.
- The events are posted complete by the system when all I/O is completed, temporary errors corrected, and length checking performed. The DECB is not checked for errors or exceptional conditions, nor are end-of-volume procedures initiated. Your program must perform these operations.

  If you coded MULTACC on the DCBE macro with a nonzero value and you issue a WAIT macro for a BSAM or BPAM DECB, then issue a TRUNC macro before the WAIT and after the previous READ or WRITE to the DECB.

Processing PDSEs and Compressed Format Data Sets

If the PDSE member is open for update or a compressed format data set is open for output, and in a storage class with "Guaranteed Synchronous Write" specified, issue a CHECK macro following a WRITE macro to guarantee that the data is synchronized to DASD. Otherwise, synchronization is not guaranteed until CLOSE, or the STOW macro or the SYNCDEV macro is issued. Synchronization occurs at CLOSE if BSAM or QSAM are used to process the PDSE members or compressed format data set. Specifying "Guaranteed Synchronous Write" in the storage class produces the same result as issuing the SYNCDEV macro.

The format of the WAIT macro is:

[*label*]	WAIT	[*number of events*] {,ECB=*addr*|ECBLIST=*addr*} [,LONG={YES|NO}]

number of events
specifies a decimal integer from 0 to 255. Zero is an effective NOP instruction; 1 is assumed if the parameter is omitted. The *number of events* must not exceed the number of event control blocks. You can also use register notation (2-12).

ECB=*addr*
specifies the address of the event control block (or DECB) representing the single event that must occur before processing can continue. The parameter is valid only if the *number of events* is specified as 1 or is omitted.

addr
specify RX type or use register notation (1-12).

ECBLIST=*addr*
specifies the address of a virtual storage area containing one or more consecutive fullwords on a fullword boundary. Each fullword contains the address of an event control block (or DECB). The high-order bit in the last word (address) must be set to 1 to indicate the end of the list. The number of event control blocks must be equal to or greater than the specified *number of events.*

LONG=[YES|NO]
specifies whether the task is entering a long wait or a regular wait. Normally, I/O events should not be considered 'long' unless it is anticipated that operator intervention is required.

Caution: A job step with all its tasks in a WAIT condition terminates on expiration of the time limits that apply to it.

Access method ECBs are maintained entirely by the access methods and supporting control program facilities. You can inspect access method ECBs, but should never modify them.

WRITE—Write a block (BDAM)

The WRITE macro adds or replaces a block in an existing direct data set. (This version of the WRITE macro cannot be used to create a direct data set because no capacity record facilities are provided.) Control might be returned to the problem program before the block is written. The output operation must be tested for completion using a CHECK or WAIT macro. A data event control block, shown in "Status information following an input/output operation" on page 373, is constructed as part of the macro expansion.

The WRITE macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The standard form of the WRITE macro is written as follows (the list and execute forms are shown following the descriptions of the standard form):

[*label*]	WRITE	*decb name* ,{DA[F]} {DI[F|X]} {DK[F|X]} ,*dcb address* ,{*area address*|'S'} ,{*length*|'S'} ,{*key address*|'S'|0} ,*block address*

***decb name*—symbol**
specifies the name that is assigned to the data event control block that is created as part of the macro expansion.

***type*—{DA[F]}**
{DI[F|X]}
{DK[F|X]}
is coded in one of the combinations shown to specify the type of write operation and optional services performed by the system:

DA
specifies that a new block is added to the data set. The search for available space starts on the track indicated by the *block address*. Fixed-length records (with keys only) are added to a data set by replacing dummy records. Variable-length records (with or without keys) are added to a data set by using available space on a track. (For more information on adding records to a direct data set, see *z/OS DFSMS Using Data Sets*. For a description of adding records with extended search, see the LIMCT parameter of the DCB macro.)

DI
specifies that a data block and key, if any, are written at the device address indicated in the area specified in the *block address*. Any attempt to write a capacity record (R0) is an invalid request when relative track addressing or actual track addressing are used, but when relative block addressing is used, relative block 0 is the first data block in the data set.

DK
specifies that a data block (only) is written using the key in the area specified by the *key address* as a search argument. The search for the block starts at the device address indicated in the area specified in the *block address*. The description of the DCB macro LIMCT parameter contains a description of the search.

F
requests that the system provide block position feedback into the area specified in the *block address*. This character can be coded as a suffix to DA, DI, or DK as shown above.

X
requests that the system release the exclusive control requested by a previous READ macro and provide block position feedback into the area specified in the *block address*. This character can be coded as a suffix to DI or DK as shown above.

***dcb address*—A-Type Address or (2-12)**
specifies the address of the data control block for the opened direct data set. When issued in 31-bit addressing mode, the input DCB address and area address must be clean 31-bit addresses.

***area address*—A-Type Address, (2-12), or 'S'**
specifies the address of the area containing the data block to be written. 'S' can be coded instead of an *area address* only if the data block (or key and data) are contained in a buffer provided by dynamic buffering. That is, 'S' was coded in the *area address* of the associated READ macro. If 'S' is coded in the WRITE macro, the *area address* from the READ macro data event control block must be moved into the WRITE macro data event control block.The buffer area acquired by dynamic buffering is released after the WRITE macro is executed. For a description of the data event control block, see "Status information following an input/output operation" on page 373. If the input area address resides above the 16MB line, you must issue the WRITE in 31-bit mode.

***length*—symbol, decimal digit, absexp, (2-12) or 'S'**
specifies the number of data bytes to be written (up to a maximum of 32760). If 'S' is coded, it specifies that the system uses the value in the block size (DCBBLKSI) field of the DCB as the length. When undefined-length records are used, if the WRITE macro is for update and the length specified differs from the original block, the new block is truncated or padded with binary zeros accordingly. The problem program can check for this situation in the SYNAD routine.

If *length* is omitted for format-U records, no error indication is given when the program is assembled, but the problem program must insert a length into the data event control block before the WRITE macro is executed.

If undefined-length records are being processed and the DCBESLB1 is set on, then 'S' must be coded. In this case, the READ length is taken from the DCBE blocksize field (DCBEBLKSI).

***key address*—A-Type Address, (2-12), 'S', or 0**
specifies the address of the area containing the key to be used. Specify 'S' instead of an address only if the key is contained in an area acquired by dynamic buffering. If the key is not to be written or used as a search argument, specify zero instead of a *key address*.

***block address*—A-Type Address or (2-12)**
specifies the address of the area containing the relative block address, relative track address, or actual device address that is used in the output operation. The length of the area depends on the type of addressing that issued and if the feedback option (OPTCD=F) is specified in the data control block.

If OPTCD=F is specified in the DCB macro and F or X is specified in the WRITE macro, you must provide a relative *block address* in the form specified by OPTCD in the DCB macro. For example, if OPTCD=R is specified, you must provide a 3-byte relative block address. If OPTCD=A is specified, you must provide an 8-byte actual device address (MBBCCHHR). If neither is specified, you must provide a 3-byte relative address (TTR).

If OPTCD=F is not specified in the DCB macro and F or X is specified in the WRITE macro, then you must provide an 8-byte actual device address (MBBCCHHR) even if relative block or relative track addressing is being used.

WRITE—Write a logical record or block of records (BISAM)

The WRITE macro adds or replaces a record or replaces an updated block in an existing indexed sequential data set. Control might be returned to the problem program before the block or record is written. The output operation must be tested for completion using a WAIT or CHECK macro. A data event control block, shown in "Status information following an input/output operation" on page 373, is constructed as part of the macro expansion.

The standard form of the WRITE macro is written as follows (the list and execute forms are shown following the descriptions of the standard form):

[*label*]	WRITE	*decb name* ,{K|KN} ,*dcb address* ,{*area address*|'S'} ,{*length*|'S'} ,*key address*

***decb name*—symbol**
specifies the name that is assigned to the data event control block that is created as part of the macro expansion.

***type*—{K|KN}**
specifies the type of write operation:

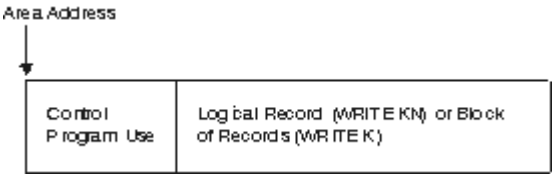
K
specifies that either an updated unblocked record or a block containing an updated record is to be written. If the record is read using a READ KU macro, the data event control block for the READ macro must be used as the data event control block for the WRITE macro, using the execute form of the WRITE macro.

KN
specifies that a new record is to be written, or a variable-length record is to be rewritten with a different length. All records or blocks of records read using READ KU macros for the same data control block must be written back before a new record can be added, except when the READ KU and WRITE KN refer to the same DECB.

***dcb address*—A-Type Address or (2-12)**
specifies the address of the data control block for the opened existing indexed sequential data set. If a block is written, the data control *block address* must be the same as the *dcb address* in the corresponding READ macro.

***area address*—A-Type Address, (2-12), or 'S'**
specifies the address of the area containing the logical record or block of records to be written. The first 16 bytes of this area are used by the system and should not contain your data. The *area address* must specify a different area than the *key address*. When new records are written (or when variable-length records arerewritten with a different length), the *area address* of the new record must always be supplied by the problem program. The addressed area might be altered by the system. 'S' can be coded instead of an address only if the block of records is contained in an area provided by dynamic buffering. That is, 'S' is coded for the *area address* in the associated READ KU macro. The addressed area is released after execution of a WRITE macro using the same DECB. The area can also be released by a FREEDBUF macro.

The following illustration shows the format of the area:



Indexed sequential buffer and work area requirements are discussed in *z/OS DFSMS Using Data Sets*.

***length*—symbol, decimal digit, absexp, (2-12) or 'S'**
specifies the number of data bytes to be written, up to a maximum of 32760. Specify 'S' unless a variable-length record is to be rewritten with a different length.

***key address*—A-Type Address or (2-12)**
specifies the address of the area containing the key of the new or updated record. The *key address* must specify a different area than the *area address*. For blocked records, this is not necessarily the high key in the block. For unblocked records, this field should not overlap with the work area specified in the MSWA of the DCB macro.

Note: When new records are written, the key area might be altered by the system.

WRITE—Write a block (BPAM and BSAM)

The WRITE macro adds or replaces a block in a sequential or partitioned data set. Control might be returned to the problem program before the block is written. The output operation must be tested for completion using the CHECK macro.

If the OPEN macro specifies UPDAT, both the READ and WRITE macros must refer to the same data event control block. See the list form of the READ or WRITE macro for a description of how to construct a data event control block. See the execute form of the READ or WRITE macro for a description of modifying an existing data event control block.

Data Conversion: You can request conversion by coding LABEL=(,AL) or (,AUL) in the DD statement, or by coding OPTCD=Q in the DCB macro or DCB subparameter of the DD statement. When conversion is requested, all records whose record format (RECFM parameter) is F, FB, D, DS, DB, DBS, or U are automatically converted from one character representation to another. Conversion is performed according to one of the following techniques:

- **Coded Character Set Identifier (CCSID) conversion**

  If CCSIDs are supplied from any source for ISO/ANSI V4 tapes, records are converted from the CCSID as seen by the problem program to the CCSID which represents the data on tape. You can also prevent conversion by supplying a special CCSID. CCSID may be supplied in the CCSID subparameter of a JOB, EXEC, or DD statement or the tape label.

- **Default Character Conversion**

  If you are using non-ISO/ANSI V4 tapes or if CCSIDs are not supplied by any source, data management converts the records from EBCDIC code to ASCII code using specific tables defined for this default character conversion.

Refer to *z/OS DFSMS Using Data Sets*, for a complete description of CCSID conversion and Default Character conversion.

If conversion from EBCDIC code to ASCII code is requested, issuing multiple WRITE macros for the same record causes an error because the first WRITE macro issued converts the output data in the output buffer into ASCII code. This problem also exists when converting from one CCSID to another.

A data event control block, shown in “Status information following an input/output operation” on page 373, is constructed as part of the macro expansion.

Processing PDSEs and Compressed Format Data Sets: If the PDSE member is open for update or a compressed format data set is open for output, and it resides ina storage class with "Guaranteed Synchronous Write" specified, issue a CHECK macro following a WRITE macro to guarantee that the data is synchronized to DASD. Otherwise, synchronization is not guaranteed until CLOSE, or the STOW macro or the SYNCDEV macro is issued. Synchronization occurs at CLOSE if BSAM or QSAM are used to process the PDSE member or compressed format data set. Specifying "Guaranteed Synchronous Write" in the storage class produces the same result as issuing the SYNCDEV macro.

When processing a compressed format data set and NOTE/POINT is specified in the DCB (MACRF=P), a WRITE issued for a block whose user RBN value exceeds 16 777 215 will result in an I/O error. This is due to the fact that the NOTE/POINT interface is limited by a 3 byte token.

z/OS UNIX files: The last write issued against a UNIX file before CLOSE denotes the end of the file. Any type of positioning (POINT, BSP, CLOSE TYPE=T REREAD) following a WRITE does not truncate the file.

Addressing mode: When you issue the WRITE macro in 24-bit mode, you provide only 24-bit addresses unless you code SF64 or SF64P. When you issue the WRITE macro in 31-bit addressing mode, all addresses must be valid 31-bit addresses unless documentation says otherwise or you code SF64 or SF64P. With SF64 or SF64P, the data area can reside above the 2 GB bar but you cannot issue WRITE in 64-bit mode.

BSAM and BPAM allow data areas to be located above the 16MB line. This includes allowing the caller to issue some other BSAM and BPAM macros in 31-bit addressing mode regardless of whether the data area is above or below the 16MB line. Most types of data sets support 31-bit mode. For more information, refer to “Environmental considerations” on page xxii.

The standard form of WRITE must be issued from a program that resides below the 16MB line because the DECB must reside below the line.

To take advantage of providing data areas above the 16MB line for BSAM macros, the issuer of the WRITE macro must execute in 31-bit addressing mode.

Syntax: The standard form of the WRITE macro is written as follows (the list and execute forms are shown following the descriptions of the standard form):

[*label*]	WRITE	*decb name* {,SF|SF64|SF64P} ,*dcb address* ,*area address* [,{*length*|'S'}]

***decb name*—symbol**
specifies the name that is assigned to the data event control block created as part of the macro expansion.

***type*—{SF|SF64|SF64P}**
is coded as shown to specify the type of write operation:

SF
specifies normal, sequential, forward operation.

SF64
for BSAM, indicates sequential forward writing and that *area address* is an 8-byte address and can point to an area above the 2-GB bar. The 8-byte pointer in the macro expansion must reside below the 2-GB bar. When the area is in a register, it must be a 64-bit register. With SF64, you code the name or address of a double word that points to the area.

If you code SF64 on the list form (MF=L), then you must code SF64P on the execute form. That means that the execute form is providing a 64-bit pointer to the data area.

The system supports this option only for extended format data sets. For other restrictions see *z/OS DFSMS Using Data Sets*.

SF64P
for BSAM, indicates sequential forward writing and that *area address* is a doubleword (8 bytes) containing the address of the area that can be above the 2-GB bar. The 8-byte pointer must reside below the 2-GB bar. With SF64, you code the name or address of the data area. With SF64P, you code the name or address of a doubleword that points to the area.

If you code SF64 on the list form (MF=L), then you must code SF64P on the execute form. That means that the execute form is providing a 64-bit pointer to the data area.

The system supports this option only for extended format data sets. For other restrictions see *z/OS DFSMS Using Data Sets*.

***dcb address*—A-Type Address, or (2-12)**
specifies the address of the data control block for the opened data set being allocated or processed. If the data set is being updated, the data control *block address* must be the same as the *dcb address* in the corresponding READ macro. When issued in 31-bit addressing mode, the input DCB address must be a clean 31-bit address. If the data area resides above the 16MB line, you must issue the WRITE in 31-bit mode.

***area address*—A-Type Address or (2-12)**
specifies the address of the problem program area in which the block is placed if you do not code SF64P. Specifies an eight-byte pointer if you specify SF64P. If you specify SF64P, the specified

doubleword contains an area pointer that can point above the 2 GB bar. The doubleword must reside below the 2 GB bar. If you specify the register form with SF64, it is a 64–bit register. Even if you specify SF64 or SF64P, your program must run in 24–bit or 31–bit mode. If a key is written (KEYLEN value is not zero), the key must precede the data in the same area. If the data area (or eight byte pointer with SF64P) resides above the 16 MB line, you must issue the WRITE in 31–bit mode.

***length*—symbol, decimal digit, absexp, (2-12) or 'S'**
specifies the number of bytes to be written. The maximum value is BLKSIZE in the DCB (without LBI) or DCBE (with LBI) but on the WRITE macro the value also must be no more than 32760. This parameter is meaningful only for undefined-length records (RECFM=U) or for ASCII records (RECFM=D) when the DCB BUFOFF is not L. If you code `'S'` to indicate the length of the block to be written, it means the length is in the data control block if you are not using LBI or in the DCBE if you are using LBI. If you are using LBI when writing format-U records or format-D records without BUFOFF=L , then you must code the `'S'` value. Omit the *length* parameter for all record formats except format-U and format-D (when BUFOFF is not L).

If *length* is omitted for format-U or format-D (with BUFOFF that is not L) records, no error indication is given when the program is assembled, but the problem program must insert a length into the data event control block before the WRITE macro is issued.

WRITE—Write a block (create a direct data set with BSAM)

The WRITE macro adds a block to the direct data set being created. For fixed-length blocks, the system writes the capacity record automatically when the current track is filled. For variable and undefined-length blocks, a WRITE macro must be issued for the capacity record. Control might be returned to the problem program before the block is written. The output operations must be tested for completion using a CHECK macro. A data event control block, shown in "Status information following an input/output operation" on page 373, is constructed as part of the macro expansion.

The standard form of the WRITE macro is written as follows (the list and execute forms are shown following the descriptions of the standard form):

[*label*]	WRITE	*decb name* ,{SF|SFR|SD|SZ} ,*dcb address* ,*area address* [,{*length*|'S'}] [,*next address*]

***decb name*—symbol**
specifies the name that is assigned to the data event control block that is created as part of the macro expansion.

***type*—{SF|SFR|SD|SZ}**
is coded as shown, to specify the type of write operation performed by the system:

SF
specifies that a new data block is written in the data set.

SFR
specifies that a new variable-length spanned record is written in the data set, and next address feedback is requested.This parameter can be specified only for variable-length spanned records (BFTEK=R and RECFM=VS are specified in the data set control block). If type SFR is specified, the *next address* parameter must be included.

SD
specifies that a dummy data block is written in the data set. Dummy data blocks can be written only when fixed-length records with keys are used.

SZ
specifies that a capacity record (R0) is written in the data set. Capacity records can be written only when variable-length or undefined-length records are used.

***dcb address*—A-Type Address or (2-12)**
specifies the address of the data control block opened for the data set being created. You must specify DSORG=PS (or PSU) and MACRF=WL in the DCB macro to create a direct data set.

***area address*—A-Type Address or (2-12)**
specifies the address of the area containing the data block to be added to the data set. If keys are used, the key must precede the data in the same area. For writing capacity records (SZ), *area address* is ignored and can be omitted (the system supplies the information for the capacity record). For writing dummy data blocks (SD), the area need be only large enough to hold the key plus one data byte. The system constructs a dummy key with the first byte set to all 1 bits (hexadecimal FF) and adds the block number in the first byte following the key. When a dummy block is written, a complete block is written from the area immediately following the *area address*. Therefore, *area address* plus the value specified in BLKSIZE and KEYLEN must be within the area allocated to the program writing the dummy blocks.

***length*—symbol, decimal digit, absexp, (2-12), or 'S'**
is used only when undefined-length (RECFM=U) blocks are being written. The parameter specifies the length of the block, in bytes, up to a maximum of 32760. If `'S'` is coded, it specifies that the system uses the length in the block size (DCBBLKSI) field of the data control block as the length of the block to be written.

If *length* is omitted for format-U records, no error indication is given when the program is assembled, but the problem program must insert a length into the data event control block before the WRITE is issued.

***next address*—A-Type Address or (2-12)**
specifies the address of the area where the system places the relative track address of the next record to be written. Next address feedback can be requested only when variable-length spanned records are used.

Using Variable-length Spanned Records: When variable-length spanned records are used (RECFM=VS and BFTEK=R are specified in the data control block), the system writes capacity records (R0) automatically in the following cases:

- When a record spans a track.
- When the record cannot be written completely on the current volume. In this case, all capacity records of remaining tracks on the current volume are written. Tracks not written are still counted in the search limit specified in the LIMCT parameter of the data control block.
- When the record written is the last record on the track, the remaining space on the track cannot hold more than 8 bytes of data.

WRITE completion codes—write a block (create a direct data set with BSAM)

After the write has been scheduled and control returns to your problem program, the three high-order bytes of register 15 are set to 0. The low-order byte contains a return code.

The WRITE return codes are as follows. For fixed-length records, the return codes are unpredictable.

Return Code (15)	Fixed-Length (SF or SD)	Variable or Undefined-length (SF or SFR)	Variable or Undefined-length (SZ)
00 (X'00')	Block is written. (If the previous return code was 08, a block is written only if the DD statement specifies secondary space allocation and sufficient space is available.)	Block is written. (If the previous return code was 08, a block is written only if the DD statement specifies secondary space allocation and sufficient space is available.)	Capacity record was written; another track is available.

Return Code (15)	Fixed-Length (SF or SD)	Variable or Undefined-length (SF or SFR)	Variable or Undefined-length (SZ)
04 (X'04')	Block is written, followed by a capacity record. (If the previous return code was 08, a block is written only if the DD statement specifies secondary space allocation and sufficient space is available.)	Block was not written; write a capacity record (**SZ**) to describe the current track, then reissue the WRITE macro.	—
08 (X'08')	Block is written, followed by a capacity record. The next block requires secondary space allocation.	—	Capacity record was written. The next block requires secondary space allocation. This code is not issued if the WRITE macro is issued on a one-track secondary extent.
12 (X'0C')	Block is not written; issue a CHECK macro for the previous WRITE macro, then reissue the WRITE macro.	Block is not written; issue a CHECK macro for the previous WRITE macro, then reissue the WRITE macro.	Block is not written; issue a CHECK macro for the previous WRITE macro, then reissue the WRITE macro.

WRITE–List and execute forms

WRITE—List form

The list form of the WRITE macro is used to construct a data management parameter list as a data event control block (DECB). For a description of the various fields in the DECB for each access method, see "Status information following an input/output operation" on page 373.

The description of the standard form of the WRITE macro explains the function of the parameters used for each access method, and the meaning of `'S'` when coded for the *area address*, *length*, and *key address* parameters. For each access method, `'S'` can be coded only for those parameters for which it can be coded in the standard form of the macro. The format description below indicates the optional and required parameters in the list form only, but does not indicate optional and required parameters for any specific access method.

The list form of the WRITE macro may be assembled into a program that resides above the 16MB line, but the execute form of READ or WRITE cannot use it there. You may copy it to below the 16MB line so the copy can be used, possibly in 31-bit mode.

The list form of the WRITE macro is:

[*label*]	`WRITE`	*decb name* ,*type* ,[*dcb address*] ,[*area address*|`'S'`] ,[*length*|`'S'`] ,[*key address*|`'S'`] ,[*block address*] ,[*next address*],`MF=L`

decb name—symbol

type—code one of the types shown in the standard form

dcb address—A-Type Address

area address—A-Type Address or **'S'**

length—symbol, decimal digit, absexp, or **'S'**

key address—A-Type Address or **'S'**

block address—A-Type Address

next address—A-Type Address

MF=L
specifies the WRITE macro is used to create a data event control block that is to be referred to by an execute-form instruction.

WRITE—Execute form

A remote data management parameter list (data event control block) is used in, and can be modified by, the execute form of the WRITE macro.The data event control block can be generated by the list form of either a READ or WRITE macro.

The description of the standard form of the WRITE macro explains the function of the parameters used for each access method, and the meaning of `'S'` when coded for the *area address*, *length*, and *key address* parameters. For each access method, `'S'` can be coded only for those parameters for which it can be coded in the standard form of the macro. The format description below indicates the optional and required parameters in the execute form only, but does not indicate the optional and required parameters for any specific access method.

The execute form of the WRITE macro is written as follows:

[*label*]	`WRITE`	*decb address* ,*type* ,[*dcb address*] ,[*area address*|`'S'`] ,[*length*|`'S'`] ,[*key address*|`'S'`] ,[*block address*] ,[*next address*] ,`MF=E`

decb address—RX-Type Address or (1-12). This must reside below the 16MB line.

type—code one of the types shown in the standard form

dcb address—RX-Type Address or (2-12)

area address—RX-Type Address, (2-12), or `'S'`

length—symbol, decimal digit, absexp, (2-12), or `'S'`

key address—RX-Type Address, (2-12), or `'S'`

block address—RX-Type Address or (2-12)

next address—RX-Type Address or (2-12)

MF=E
specifies that the execute form of the WRITE macro is used, and an existing data event control block (specified in the *decb address*) is to be used by the access method.

XLATE—Translate to and from ASCII (BSAM and QSAM)

The XLATE macro is used to convert the data in an area in virtual storage from ASCII code to EBCDIC code or from EBCDIC code to ASCII code.

Refer to *z/OS DFSMS Using Data Sets*, for the ASCII to EBCDIC and EBCDIC to ASCII conversion codes. When converting EBCDIC code to ASCII code, all EBCDIC code not having an ASCII equivalent is converted to X'1A'. When converting ASCII code to EBCDIC code, all ASCII code not having an EBCDIC equivalent is converted to X'3F'. Bit 0 is always set to 0 during EBCDIC to ASCII conversion and is expected to be 0 during ASCII to EBCDIC conversion.

The XLATE macro may be issued in 24- or 31-bit addressing mode. When issued in 31-bit addressing mode, all addresses must be valid 31-bit addresses.

The format of the XLATE macro is:

[*label*]	XLATE	*area address* ,*length* [,TO={A|E}]

***area address*—RX-Type Address, symbol, decimal digit, absexp, (2-12), or (1)**
specifies the address of the area to be converted. If issued in 31-bit addressing mode, this area may reside above or below the 16MB line.

***length*—symbol, decimal digit, absexp, (2-12), or (0)**
specifies the number of bytes to be converted.

TO={A|E}
specifies the type of conversion that is requested. If this parameter is omitted, E is assumed. You can specify:

A
specifies conversion from EBCDIC code to ASCII code.

E
specifies conversion from ASCII code to EBCDIC code.

The return codes are shown in Table 69 on page 369.

Table 69. XLATE Return Codes

Return Code	Meaning
0 (X'00')	Success
4 (X'04')	Zero or negative number of bytes to translate
8 (X'08')	Invalid data address

Appendix A. Macros available by access method

Macro	VSAM	BDAM	BPAM	BSAM	QSAM	Supports 31-Bit "1" on page 372
ACB	X					X
BLDL			X			X
BLDVRP	X					X
BSP			X	X		X
BUILD		X	X	X	X	X
BUILDRCD					X	X
CHECK	X	X	X	X		X
CLOSE	X	X	X	X	X	X
CNTRL				X	X	X
DCB		X	X	X	X	N/A "3" on page 372
DCBD		X	X	X	X	X
DCBE			X	X	X	X
DESERV			X			X
DLVRP	X					X
ENDREQ	X					X
ERASE	X					X
ESETL						
EXLST	X					X
FEOV				X	X	X
FIND			X			X
FREEBUF		X	X	X		X
FREEDBUF		X				X
FREEPOOL		X	X	X	X	X
GENCB	X					X
GET	X				X	X
GETBUF		X	X	X		X
GETPOOL		X	X	X	X	X
IHADCBE			X	X	X	X
ISITMGD		X	X	X	X	X
MODCB	X					X
MRKBFR	X					X
MSGDISP				X	X	X
NOTE			X	X		X
OPEN	X	X	X	X	X	X
PDAB					X	
PDABD					X	
POINT	X		X	X		X
PRTOV				X	X	X
PUT	X				X	X

Macro	VSAM	BDAM	BPAM	BSAM	QSAM	Supports 31-Bit"1" on page 372
PUTX					X	X
READ		X	X	X		X"4" on page 372
RELEX		X				X
RELSE					X	X
RPL	X					X
SCHBFR	X					X
SETL						
SETPRT				X	X	X"4" on page 372
SHOWCB	X					X
STOW			X			X
SYNADAF		X	X	X	X	X
SYNADRLS		X	X	X	X	X
SYNCDEV			X	X	X	X
TESTCB	X					X
TRUNC				X	X	X
VERIFY	X					X
WAIT		X	X	X		X
WRITE		X	X	X		X"4" on page 372
WRTBFR	X					X
XLATE				X	X	X

Notes:

1. For non-executable macros, this means it can reside above the 16 MB line. For executable macros, this indicates that the macro issuer can be in 31-bit mode. The individual macro descriptions state if certain storage must be below the 16MB line.
2. Can be issued but has no effect.
3. Non-executable macro. You can assemble the DCB macro into a program that resides above the 16MB line, but the program must move it below the line before using it. Except for the DCBE, all areas that the DCB refers to, such as EXLST, SYNAD, and EODAD, must be below the 16MB line.
4. The list form of the READ, WRITE, and SETPRT macro can be assembled into a program that resides above the 16MB line, but the execute form of the macro cannot use it there. You can copy it to below the 16MB line so the copy can be used, possibly in 31-bit mode. Do not issue the standard form of the macro in a program that resides above the 16MB line.

Appendix B. Non-VSAM control blocks

This section discusses:

- The format of the DECB which shows the status of the I/O operation.
- Data control block symbolic field names.
- Data control block extension (DCBE).

Status information following an input/output operation

Following an I/O operation with a DCB, the control program makes certain status information available to the problem program. This information is a 2-byte exception code or an area of standard status indicators, or both.

Exception codes are provided in the ISAM data control block (DCB) or in the BDAM data event control block (DECB). The DECB is described below, and the exception code is within the block. If you code a DCBD macro, you can address the exception code in a data control block as two 1-byte fields, DCBEXCD1 and DCBEXCD2.

Status indicators are available only to the error analysis routine designated by the SYNAD entry in the data control block (DCB) or data control block extension (DCBE). A pointer to the status indicators is provided either in the DECB (for BSAM, BPAM, and BDAM) or in register 0 (for QISAM and QSAM). For more information on exception codes and status indicators, see *z/OS DFSMS Using Data Sets*.

Data event control block

A DECB is constructed as part of the expansion of READ and WRITE macros and is used to pass parameters to the control program, help control the read or write operation, and receive indications of the success or failure of the operation. The DECB is named by the READ or WRITE macro, begins on a fullword boundary, resides below the 16MB line, and contains the information shown in the following illustration:

Offset from DECB Address (Bytes)	Field Contents BSAM and BPAM	Field Contents BISAM	Field Contents BDAM
0	ECB	ECB	ECB[“1” on page 373]
+4	Type	Type	Type
+6	Length	Length	Length
+8	DCB address	DCB address	DCB address
+12	Area address	Area address	Area address
+16	Address of status indicators. Status indicators reside below the 16MB line.	Logical record address	Address of status indicators. Status indicators reside below the 16MB line.
+20		Key address	Key address
+24		Exception code (2 bytes)	Block address
+28			Next address

Note:

1. The control program returns exception codes in bytes +1 and +2 of the ECB.

Data control block symbolic field names

The following describes data control block fields containing information that defines the data characteristics and device requirements for a data set. Each of the fields described shows the values that result from specifying various options in the DCB macro. These fields can be referred to by the problem program by a DCBD macro that creates a dummy control section (DSECT) for the data control block. Fields that contain addresses are 4 bytes long and are aligned on a fullword boundary. If the problem program inserts an address into a field, the address must be inserted into the low-order 3 bytes of the field without changing the high-order byte.

The contents of some fields in the data control block depend on the device and access method being used. A separate description is provided when the contents of the field are not common to all device types and access methods.

For diagnosis purposes, *z/OS DFSMSdfp Diagnosis* describes more fields.

Data control block—common fields

Offset	Bytes in Length	Field Name	Description
26(1A)	2	DCBDSORG	Data set organization. **Code** 1000 000x **IS** Indexed sequential. 0100 000x **PS** Physical sequential. 0010 000x **DA** Direct organization. ...x xx.. Reserved bits. 0000 001x **PO** Partitioned organization. ....1 **U** Unmovable—the data set contains location-dependent information.
40(28)	8	DCBDDNAM	Eight-byte name of the data definition statement that defines the data set associated with this DCB. (Before DCB is opened.)
40(28)	2	DCBTIOT	(After DCB is opened.) Offset from the TIOT origin to the TIOELNGH field in the TIOT entry for the DD statement associated with this DCB. Unsigned. Maximum value is just below 64K.
42(2A)	2	DCBMACRF	This field can only be referred to during and after OPEN. It is common to all uses of the DCB and is created by moving the DCBMACR field into this area.
45(2D)	3	DCBDEBA	(After DCB is opened.) Address of field, DEBBASIC, in the associated DEB.
48(30)	1	DCBOFLGS	Flags used by open routine.

Offset	Bytes in Length	Field Name	Description
		...1	OPEN has completed successfully.
		.... 1...	Set to 1 by problem program to indicate concatenation of unlike attributes.
		.... ..0.	Set to 0 by an I/O support function when that function takes a user exit. It is set to 0 to inhibit other I/O support functions from processing this DCB.
		.... ..1.	Set to 1 on return from the I/O support function that took the exit.
50(32)	2	DCBMACR (Before OPEN)	Macro reference before OPEN. Major macros and various options associated with them. Used by the open routine to determine access method. Used by the access method executed with other parameters to determine which load modules are required. This field is moved to overlay part of DCBDDNAM at OPEN time and becomes the DCBMACRF field. This field is common to all uses of the DCB, but each access method must be referenced for its meaning. For EXCP, bit 0 is always on. If not EXCP, then bit 0 is off and exactly one of the next two bits is on.

Data control block—BPAM, BSAM, QSAM

Offset	Bytes in Length	Field Name	Description
20(14)	4	DCBBUFCB	Address of buffer pool control block.
20(14)	1	DCBBUFNO	Number of buffers required for this data set. Can range from 0 to 255. Default = 1 for PDSEs; various defaults
21(15)	3	DCBBUFCA	Address of buffer pool control block. If the low-order bit is 1, this field does not point to a buffer pool.
24(18)	2	DCBBUFL	Length of buffer. Can range from 0 to 32760 bytes. Is based on BLKSIZE.
32(20)	1	DCBHIAR	
		0... .0..	No DCBE, no HIARCHY
		1... .0..	No DCBE, HIARCHY=1
		0... .1..	No DCBE, HIARCHY=0
		1... .1..	DCBDCBE points to DCBE, no HIARCHY
32(20)	1	DCBBFALN	Buffer alignment: Code
		xx	Reserved bits.
		10	**D** Doubleword boundary.

Offset	Bytes in Length	Field Name	Description
		01	**F** Fullword not a doubleword boundary, coded in the DCB macro.
32(20)	1	DCBBFTEK	Buffering technique: Code
		.xxx	Reserved bits.
		.100	**S** Simple buffering.
		.110	**A** QSAM locate mode processing of spanned records: OPEN is to construct a record area if it automatically builds buffers.
		.010	**R** BSAM create BDAM processing of unblocked spanned records: Software track overflow. OPEN forms a segment work area pool. However, WRITE uses a segment work area to write a record as one or more segments. BSAM input processing of unblocked spanned records with keys: Record offset processing. READ reads one record segment into the record area. The first segment of a record is preceded in the record area by the key. Subsequent segments are at an offset equal to the key length.
		.... 1...	XLRI being used to process a RECFM=DS or RECFM=DBS format tape data set (QSAM).
33(21)	3	DCBEODA	End-of-data address. Address of a user-provided routine to handle end-of-data conditions. If the low-order bit is 1, this field does not point to an end-of-data address.
36(24)	1	DCBRECFM	Record format. Code
		000.	Record format not available.
		001.	**D** Format-D record.
		10.. ...	**F** Fixed record length.
		01..	**V** Variable record length.
		11..	**U** Undefined record length.
		..1.	**T** Track overflow.

Offset	Bytes in Length	Field Name	Description
		...1	**B** Blocked records. Cannot occur with undefined (U).
		.... 1....	**S** Fixed length record format: Standard blocks. (No truncated blocks or unfilled tracks are embedded in the data set.) Variable length record format: Spanned records.
		.... .10.	**A** ISO/ANSI control character at the beginning of each record.
		.... .01.	**M** Machine control character at the beginning of each record.
		.... .00.	No control character.
		1	Key length (KEYLEN) was specified in the DCB macro. This bit is inspected by the open routine to prevent overriding a specification of KEYLEN=0 by a nonzero specification in the JFCB or data set label.
37(25)	3	DCBEXLSA	Exit list. Address of a user-provided exit list control block.
42(2A)	2	DCBMACRF	Macro reference after OPEN. Contents and meaning are the same as those of the DCBMACR field in the foundation segment before OPEN.
50(32)	2	DCBMACR (Before OPEN)	Major macros and various options associated with them. Used by the open routine to determine access method. Code
		Byte 1	BSAM—Input
		00..	Always zero for BSAM.
		..1.	**R** READ
		.... .1..	**P** POINT (which implies NOTE).
		.... ..1.	**C** CNTRL
		...x x..x	Reserved.
51(33)		**Byte 2**	BSAM—Output
		00..	Always zero for BSAM.

Offset	Bytes in Length	Field Name	Description
		..1.	**W** WRITE
		.... 1...	**L** Load mode BSAM (create direct data set).
		.... .1..	**P** POINT
		.... ..1.	**C** CNTRL
		1	BSAM create BDAM processing of unblocked spanned records, with BFTEK=R specified: The user's program has provided a segment work area pool.
		...x	Reserved.
50(32)		Byte 1	QSAM—Input
		0...	Always zero for QSAM.
		.1..	**G** GET
		..0.	Always zero for QSAM.
		...1	**M** Move mode.
		.... 1...	**L** Locate mode.
		.... ..1.	**C** CNTRL
		1	**D** Data mode.
		.... .x..	Reserved.
51(33)		Byte 2	QSAM—Output
		0...	Always zero for QSAM.
		.1..	**P** PUT

Offset	Bytes in Length	Field Name	Description
		..0.	Always zero for QSAM.
		...1	**M** Move mode.
		.... 1...	**L** Locate mode.
		.... ..1.	**C** CNTRL
		1	**D** Data mode.
		.... .x..	Reserved.
50(32)		Byte 1	BPAM—Input
		00..	Always zero for BPAM.
		..1.	**R** READ
		.... .1..	**P** POINT (which implies NOTE).
		...x x.xx	Reserved bits.
51(33)		Byte 2	BPAM—Output
		00..	Always zero for BPAM.
		..1.	**W** WRITE
		.... .1..	**P** POINT (which implies NOTE).
		...x x.xx	Reserved bits.
52(34)	1	DCBOPTCD	Option codes.
			Code
		1...	**W** Write-validity check (DASD).

Offset	Bytes in Length	Field Name	Description
		.1..	**U** Allow a data check caused by a character that is not valid. (Impact printer with UCS feature.) Write-tape-immediate mode (3480 and 3490).
			B Treat EOF and EOV labels as EOV labels which allows SL or AL tapes to be read out of order. (Magnetic tape.)
		..1.	**C** Chained scheduling requested.
		...1	**H** Input Tape Files: Requests the testing for and bypassing of any embedded VSE checkpoint records found. (This code can only be specified in a JCL statement.)
		.... 1...	**Q** An ASCII data set is to be processed. The tape does not have to have ISO/ANSI labels.
		.... .1..	**Z** Magnetic tape devices: Use reduced error recovery procedure.
		.... ..1.	**T** BSAM and QSAM only: user totaling.
		1	**J** Specifies that the first data byte in the output data line will be a 3800 table reference character for dynamic selection of character sets.
57(39)	3	DCBSYNA	Address of user's synchronous error routine to be entered when a permanent error occurs. If the low-order bit is 1, this field does not point to an error routine.
62(3E)	2	DCBBLKSI	Block size.

Access method interface

BSAM, BPAM interface

Offset	Bytes in Length	Field Name	Description
61(3D)	1	DCBCIND2	Condition indicators.
		.... .1..	DCBCNCHS. Chain scheduling being supported. Set in OPEN. Zero for DASD. May differ from OPTCD=C bit(DCBOPTC).
72(48)	1	DCBNCP	Number of channel programs. Number of READ or WRITE requests that can be issued before a CHECK. Maximum number: 255.
80(50)	1	DCBUSASI/ DCBLBP	ASCII tape. Block prefix.

Offset	Bytes in Length	Field Name	Description
		.1..	Block prefix is a 4-byte field containing the block length.
81(51)	1	DCBBUFOF	Block prefix length.
82(52)	2	DCBLRECL	Logical record length. For fixed-length blocked record format, the presence of DCBLRECL allows BSAM to read truncated records. For undefined records, this field contains block size.

QSAM interface

Offset	Bytes in Length	Field Name	Description
61(3D)	1	DCBCIND2	Condition indicators.
		.... .1..	DCBCNCHS. Chain scheduling being supported. Set in OPEN. Zero for DASD. May differ from OPTCD=C bit(DCBOPTC).
68(44)	4	DCBIOBA	Address of area to determine length of undefined-length record after a GET macro with LBI.
80(50)	1	DCBUSASI/ DCBQSWS	ASCII tape.
		.1..	Block prefix is a 4-byte field containing the block length. (BUFOFF=L was specified).
		.... .1..	DCBOPEN. QSAM parallel input processing.
81(51)	1	DCBBUFOF	Block prefix length.

Offset	Bytes in Length	Field Name	Description
82(52)	2	DCBLRECL	Format-F records: Record length. Format-U records: Block size. Format-V records: • Unspanned record format: GET: PUTX; record length. PUT: Actual or maximum record length. • Spanned record format: Locate mode: –GET: Segment length. –PUT: Actual or minimum segment length. Logical record interface: – Before OPEN: Maximum logical record length. – After GET: Record length. – Before PUT: Actual or maximum record length. – ISO/ANSI spanned record format with XLRI; length of the record area in 'K' units (1024). Move mode: – GET: Record length. – PUT: Actual or maximum record length. • Data mode, GET: Data records up to 32752 bytes: Data length. Data records exceeding 32752 bytes: – Before OPEN: X'8000' – After OPEN: Data length. • Output mode, PUTX (output data set): Segment length.
84(54)	1	DCBEROPT	Error option. Disposition of permanent errors if the user returns from a synchronous error exit (DCBSYNAD), or if the user has no synchronous error exit.
		100.	ACC: Accept.
		010.	SKP: Skip.
		001.	ABE: Abnormal end of task.
		...x xxxx	Reserved bits.
90(5A)	2	DCBPRECL	Block length, maximum block length, or data length. Not part of LBI.

Direct access storage device interface

Offset	Bytes in Length	Field Name	Description
0(0)	4	DCBRELAD, DCBDCBE	For partitioned data sets: DCBRELAD is TTRN (beginning address) of a member. If the DCBHIAR bits at DCB offset 32 both are on, this word points to the DCBE. If the DCBE exists, and the data set is partitioned, member address is in DCBERELA in the DCBE.
4(4)	1	DCBKEYCN	Keyed block overhead.

Offset	Bytes in Length	Field Name	Description
5(5)	8	DCBFDAD	Direct access address.
16(10)	1	DCBKEYLE	Key length of the data set.
17(11)	1	DCBDEVT	Device type.
		0010	Class of device. This code means DASD
		 xxxx	Type of DASD. Programs that do not test this byte run in more environments. If a program tests this byte, it is best to test only the first four bits (class).
18(12)	2	DCBTRBAL	Current track balance. For TRKCALC macro. Not recommended for arithmetic calculation.

Magnetic tape interface

Offset	Bytes in Length	Field Name	Description
12(0C)	4	DCBBLKCT	Number of blocks in the file on the current volume to the current position.
16(10)	1	DCBTRTCH	Tape recording technique for 7-track tape.
			Code
		0010 0011	**E** Even parity.
		0011 1011	**T** BCD/EBCDIC conversion.
		0001 0011	**C** Data conversion.
		0010 1011	**ET** Even parity and conversion.
			Tape recording technique for a magnetic tape subsystem with Improved Data Recording Capability. Use TRTCH to override the system default value.
		0000 1000	**COMP** Record data in compacted format.
		0000 0100	**NOCOMP** Record data in standard format.
17(11)	1	DCBDEVT	Device type.
		1000	Class of device. This code means magnetic tape.
		 xxxx	Type of magnetic tape.
18(12)	1	DCBDEN	Tape density—3400 series magnetic tape units.
			Code
		0100 0011	**1** 556 BPI (7-track) N/A (9-track) N/A (18-track)

Offset	Bytes in Length	Field Name	Description
		1000 0011	**2** 800 BPI (7-track) 800 (9-track) N/A (18-track)
		1100 0011	**3** N/A BPI (7-track) 1600 (9-track) N/A (18-track)
		1101 0011	**4** N/A BPI (7-track) 6250 (9-track) N/A (18-track)

Card reader, card punch interface

Offset	Bytes in Length	Field Name	Description
16(10)	1	DCBMODE, DCBSTACK	
			Code
		1000	**C** Column binary mode.
		0100	**E** EBCDIC mode.
		 xxxx	Stacker selection.
		 0001	**1** Stacker 1.
		 0010	**2** Stacker 2.
		 0011	**3** Stacker 3.
17(11)	1	DCBDEVT	Device type.
		0100	Device class is unit record or TSO terminal.
		0100 0001	2540 Card Reader
		0100 0010	2540 Card Punch
		0100 0100	2501 Card Reader
		0100 0110	3505 Card Reader
		0100 1100	3525 Card Punch

Printer interface

Offset	Bytes in Length	Field Name	Description
16(10)	1	DCBPRTSP	Number indicating normal printer spacing.
			Code

Offset	Bytes in Length	Field Name	Description
		0000 0000	**0** No spacing.
		0000 0001	**1** Space one line.
		0001 0001	**2** Space two lines.
		0001 1001	**3** Space three lines.
17(11)	1	DCBDEVT	Device type.
		0100	Device class is unit record or TSO terminal.
		0100 1000	1403 Printer
		0100 1001	3211 Printer
		0100 1011	3203 Printer
		0100 1101	Look at UCBTYP field or issue the DEVTYPE macro for the actual type of printer.
		0100 1110	3800 Printing Subsystem
18(12)	1	DCBPRTOV	Test-for-printer-overflow mask (PRTOV mask). If printer overflow is to be tested for, the PRTOV macro sets the mask as follows:
			Mask
		0010 0000	**9** Test for channel 9 overflow.
		0001 0000	**12** Test for channel 12 overflow.
19(13)	1	DCBPRBYT	
		xxxx xx..	Reserved.
		11	Bits to identify currently active table reference character when 3800 printer is operating under OPTCD=J.

TSO terminal interface

Offset	Bytes in Length	Field Name	Description
17(11)	1	DCBDEVT	Device type.
		X'4F'	Device type and class are a TSO terminal (TERM=TS on the DD statement)

Data control block—ISAM

Offset	Bytes in Length	Field Name	Description
16(10)	1	DCBKEYLE	Key length.
17(11)	1	DCBDEVT	Device type.
20(14)	1	DCBBUFNO	Number of buffers required for this data set: 0-255.
21(15)	3	DCBBUFCA	Address of buffer pool control block.
24(18)	2	DCBBUFL	Length of buffer: 0 - 32760 bytes.
32(20)	1	DCBBFALN	Buffer alignment
			Code
		xx	Reserved bits.
		10	**D** Doubleword boundary.
		01	**F** Fullword not a doubleword boundary, coded in the DCB macro.
		11	**F** Fullword not a doubleword boundary, coded in the DD statement.
33(21)	3	DCBEODA	Address of a user-provided routine to handle end-of-data conditions.
36(24)	1	DCBRECFM	Record format.
			Code
		10..	**F** Fixed length records.
		01..	**V** Variable length records.
		11..	**U** Undefined length records.
		..1.	**T** Track overflow.
		...1	**B** Blocked records. Cannot occur with undefined (U).
		.... 1...	**S** Standard records. No truncated blocks or unfilled tracks are embedded in the data set.
		.... .10.	**A** ISO/ANSI control character.
		.... .01.	**M** Machine control character.

Offset	Bytes in Length	Field Name	Description
		.... .00.	No control character.
		1	Key length (KEYLEN) was specified in the DCB macro; this bit is inspected by the open routine to prevent overriding a specification of KEYLEN=0 by a nonzero specification in the JFCB or data set label.
37(25)	3	DCBEXLSA	Exit list. Address of a user-provided list.
42(2A)	2	DCBMACRF	Macro reference after OPEN: Contents and meaning are the same as those of the DCBMACR field before OPEN.
50(32)	2	DCBMACR	Macro reference before OPEN: specifies the major macros and various options associated with them. Used by the open routine to determine access method. Used by the access method executors with other parameters to determine which load modules are required. Code
50(32)		Byte 1	BISAM
		00.0 0...	Always zero for BISAM.
		..1.	**R** READ
		.... .1..	**S** Dynamic buffering.
		.... ..1.	**C** CHECK
		x	Reserved bit.
51(33)		Byte 2	BISAM
		00.0 0000	Always zero for BISAM.
		..1.	**W** WRITE
50(32)		Byte 1	QISAM
		0.0. .0..	Always zero for BISAM.
		.1..	**G** GET

Offset	Bytes in Length	Field Name	Description
		...1	**M** Move mode of GET.
		.... 1...	**L** Locate mode for GET.
		xx	Reserved bit.
51(33)		Byte 2	QISAM
		1...	**S** SETL
		.1..	**P** PUT or PUTX
		..0.	Always zero for QISAM.
		...1	**M** Move mode of PUT.
		.... 1...	**L** Locate mode for PUT.
		.... .1..	**U** Update in place (PUTX).
		.... ..1.	**K** SETL by key.
		1	**I** SETL by ID.
52(34)	1	DCBOPTCD	Option codes: Code
		1...	**W** Write-validity check.
		.1..	**U** Full-track index write.
		..1.	**M** Master indexes.
		...1	**I** Independent overflow area.
		.... 1...	**Y** Cylinder overflow area.
		.... ..1.	**L** Delete option.

Offset	Bytes in Length	Field Name	Description
		1	**R** Reorganization criteria.
		.... .x..	Reserved bit.
53(35)	1	DCBMAC	Extension of the DCBMACRF field for ISAM. Code
		xxxx ...x	Reserved bits.
		.... 1...	**U** Update for read.
		.... .1..	**U** Update type of write.
		.... ..1.	**A** Add type of write.
54(36)	1	DCBNTM	Number of tracks that determines the development of a master index. Maximum permissible value: 99.
55(37)	1	DCBCYLOF	The number of tracks to be reserved on each prime data cylinder for records that overflow from other tracks on that cylinder. To determine how to calculate the maximum number, see the section on allocating space for an indexed sequential data set in *z/OS DFSMS Using Data Sets*.
56(38)	4	DCBSYNAD	Address of user's synchronous error routine to be entered when uncorrectable errors are detected in processing data records.
60(3C)	2	DCBRKP	Relative position of the first byte of the key in each logical record. Maximum permissible value: logical record length minus key length.
62(3E)	2	DCBBLKSI	Block size.
64(40)	4	DCBMSWA	Address of the storage work area reserved for use by the control program when new records are being added to an existing data set. The DCBMSWA field contains significant information only when the data set is opened for BISAM.
68(44)	2	DCBSMSI	Number of bytes in area reserved to hold the highest level index. The DCBSMSI field contains significant information only when the data set is opened for BISAM.
70(46)	2	DCBSMSW	Number of bytes in work area used by control program when new records are being added to the data set. The DCBSMSW field contains significant information only when the data set is opened for BISAM.
72(48)	1	DCBNCP	Number of copies of the READ-WRITE (type K) channel programs that are to be established for this data control block (99 maximum).
73(49)	3	DCBMSHIA	Address of the storage area holding the highest level index.

Offset	Bytes in Length	Field Name	Description
80(50)	1	DCBEXCD1	First byte in which exceptional conditions detected in processing data records are reported to the user.
		1...	Lower key limit not found.
		.1..	Invalid device address for lower limit (QISAM only). Record length check (BISAM only).
		..1.	Space not found.
		...1	Invalid request.
		.... 1...	Uncorrectable input error.
		.... .1..	Uncorrectable output error (BISAM only). Block could not be reached (BISAM only).
		.... ..1.	Block could not be reached (input) (QISAM only). Overflow record (BISAM only).
		1	Block could not be reached (update) (QISAM only). Duplicate record (BISAM only).
81(51)	1	DCBEXCD2	Second byte in which exceptional conditions detected in processing data records are reported to the user (QISAM only).
		1...	Sequence check.
		.1..	Duplicate record.
		..1.	DCB closed when error was detected.
		...1	Overflow record.
		.... 1...	PUT: length field of record larger than length indicated in DCBLRECL.
		xxx	Reserved bits.
82(52)	2	DCBLRECL	Logical record length for fixed-length record formats. Variable-length record formats: maximum logical record length or an actual logical record length changed dynamically by the user when creating the data set.
150(96)	2	DCBNCRHI	Number of storage locations needed to hold the highest level index.
197(C5)	1	DCBOVDEV	Device type for independent overflow.

Data control block—BDAM

Offset	Bytes in Length	Field Name	Description
16(10)	1	DCBKEYLE	Key length.
17(11)	3	DCBREL	Maximum number of tracks or blocks based on the amount of space allocated for this data set.
20(14)	1	DCBBUFNO	Number of buffers required for this data set. Can range from 0 to 255.
21(15)	3	DCBBUFCA	Address of buffer pool control block or of dynamic buffer pool control block.

Offset	Bytes in Length	Field Name	Description
24(18)	2	DCBBUFL	Length of buffer. Can range from 0 to 32760.
32(20)	1	DCBBFALN	Buffer alignment:
		10	Doubleword boundary.
		01	Fullword not a doubleword boundary, coded in the DCB macro.
		11	Fullword not a doubleword boundary, coded in the DD statement.
		.x.x x...	Reserved bits.
32(20)	1	DCBBFTEK	Buffering technique.
			Code
		..1.	**R** Unblocked spanned records: Variable spanned record format. Open forms a segment work area pool. The number of segment work areas is determined by DCBBUFNO. WRITE uses a segment work area to write a record as one or more segments. READ uses a segment work area to read a record that was written as one or more segments.
36(24)	1	DCBRECFM	Record format.
			Code
		10..	**F** Fixed record length.
		01..	**V** Variable record length.
		11..	**U** Undefined record length.
		..1.	**T** Track overflow.
		...1	**B** Blocked (allowed only with V).
		.... 1...	**S** Spanned (allowed only with V).
		.... .00.	Always zeros.
		1	Key length (KEYLEN) was specified in the DCB macro. This bit is inspected by the open routine to prevent overriding a specification of KEYLEN=0 by a nonzero specification in the JFCB or data set label.
37(25)	3	DCBEXLSA	Exit list. Address of a user-provided exit list control block.

Offset	Bytes in Length	Field Name	Description
42(2A)	2	DCBMACRF	Macro reference after OPEN. Contents and meaning are the same as DCBMACR before OPEN.
50(32)	2	DCBMACR	Macro reference before OPEN: major macros and various options associated with them that will be used.
50(32)		Byte 1	Code
		00..	Always zero for BDAM.
		..1.	**R** READ
		...1	**K** Key segment with READ.
		.... 1...	**I** ID argument with READ.
		.... .1..	**S** System provides area for READ (dynamic buffering).
		.... ..1.	**X** Read exclusive.
		1	**C** CHECK macro.
51(33)		Byte 2	Code
		00..	Always zero for BDAM.
		..1.	**W** WRITE
		...1	**K** Key segment with WRITE.
		.... 1...	**I** ID argument with WRITE.
		.... .x..	Reserved bit.
		.... ..1.	**A** Add type of WRITE.
		1	Unblocked spanned records, with BFTEK=R specified and no dynamic buffering: The user's program has provided a segment work area pool.
52(34)	1	DCBOPTCD	Option codes: Code

Offset	Bytes in Length	Field Name	Description
		1...	**W** Write-validity check.
		.1..	Track overflow.
		..1.	**E** Extended search.
		...1	**F** Feedback.
		.... 1...	**A** Actual addressing.
		.... .1..	Dynamic buffering.
		.... ..1.	Read exclusive.
		1	**R** Relative block addressing.
56(38)	4	DCBSYNAD	Address of SYNAD (synchronous error) routine.
62(3E)	2	DCBBLKSI	Block size.
81(51)	3	DCBLIMCT	Number of tracks or number of relative blocks to be searched (extended search option).

Data control block extension (DCBE)

A DCBE is defined by a DCBE macro and mapped by an IHADCBE macro. The IHADCBE macro has no options. It always generates the DCBE DSECT.

Offset	Length or Bit Pattern	Field Name	Description
0(0)		DCBE	DSECT name.
0(0)	4	DCBEID	DCBE eyecatcher 'DCBE'
4(4)	2	DCBELEN	Length of DCBE. Minimum value is 56. The current level of the system ignores fields beyond that offset; however, future levels may use those fields.
6(6)	2		Reserved.
8(4)	4	DCBEDCB	DCB address. Set by system. Must be zero when OPEN is issued. Set to zero at CLOSE.
12(C)	4	DCBERELA	Partitioned data set — address (in the form TTRN) of member currently used.
16(10)	1	DCBEFLG1	Flags set by system.
	1...	DCBEOPEN	DCBE has been successfully opened.

Offset	Length or Bit Pattern	Field Name	Description
	.1..	DCBEMD31	System allows user to call access method in 31-bit mode and, if QSAM, system will honor DCBEBU31. Set by system before DCB OPEN exit.
	..1.	DCBESLBI	In DCB OPEN exit routine: the system will support large block interface (LBI) if requested. After DCB OPEN exit routine: the user requested LBI (by turning on DCBEULBI) and OPEN set this bit on before the DCB OPEN exit. In this case, DCBEBLKSI contains a valid value and the user must not use DCBBLKSI.
	...1	DCBE_32BIT_INUSE	Device using 32-bit block numbers might be due to CAPACITYMODE=XCAP
	.... 1...	DCBEBENEFIX	Performance would be benefited if the application program were to set DCBEBFXU and fix all data pages. Requires authorization. Set by system before DCB OPEN exit.
17(11)	1	DCBEFLG2	Flags set by user.
	1...	DCBEBU31	RMODE31=BUFF. QSAM buffers may be above 16 MB line and CLOSE will free them. System may test this during concatenation. This will be ignored for BSAM and user supplied buffers.
	.1..	DCBENEOD	PASTEOD=YES. The system's indicator of the end of the data set is to be ignored on input for striped data sets.
	..1.	DCBE_CONCURRENTRW (CONCURRENTRW=YES)	The data set may be read at the same time it is being written.
	...1	DCBENVER	NOVER=YES. OPEN is to bypass the verification of consistent stripes of a striped data set.
	.... 1...	DCBEGSIZ	GETSIZE=YES. OPEN is to calculate the size of the data set (RBNs) and store this number in DCBESIZE and DCBEXSIZ.
	.... .1..	DCBEULBI	User requests large block interface (LBI). Before OPEN calls the DCB OPEN exit routine, a 1 means that DCBEBLKSI is valid and might be 0.
	.... ..1.	DCBE_REQST_XCAP	CAPACITYMODE=XCAP. Extended capacity. Device to use 32-bit block identifiers if supported.
	1	DCBEEXPS	Bypass extended data integrity checking. Caller must be system key, supervisor state or APF authorized for this to have effect.
18(12)	2	DCBENSTR	Number of stripes for a striped data set. Zero if data set is not striped. Set by OPEN or when switching between data sets in a concatenation. Set before OPEN or EOV exit is called.
20(14)	1	DCBEFLAG3	Flags set by user.
	1...	DCBELARGE	Indicates whether the application program can handle the interface for large format data sets. 1 means yes and 0 means no.

Offset	Length or Bit Pattern	Field Name	Description
	.1..	DCBEBFXU	FIXED=USER. Indicates no I/O pagefixing needed. User is responsible for passing fixed buffers.
	..1.	DCBEEADSCBOK	EADSCB=OK. Indicates users supports 28-bit cylinder DSCBs.
	...1	DCBELOCANY	User allows XTIOT and allows DSABs and UCBs to be in 31-bit storage.
	.... .001	DCBESYNC_SYSTEM	SYNC=SYSTEM
	.... .111	DCBESYNC_NONE	SYNC_NONE. 000 means SYNC is not specified. Combinations of these three bits other than 000, 001 and 111 are reserved.
21(15)	1	DCBEFLG5	Flags set by user.
	.... .1..	DCBEDSENCRYPTOK	DSENCRYPT=OK specified. Indicates the EXCP application supports data set encryption.
	.... ..1.	DCBEDSENCNP	Indicates the EXCP application supports accessing an encrypted basic and large format data set without prefixes. Only supported with an EXCP DCB. On first open for OUTPUT/OUTIN/INOUT/UPDAT, this indicator will be saved in the encryption cell. For a non-prefixed encrypted basic and large format data set, this flag is required for any subsequent open for reading and writing. The DCBE macro does not have a keyword for this.
	1	DCBETrackLock	CONCURRENTRW=(YES,TRKLOCK) specified. This means the application can tolerate inconsistent read data due to concurrent writes to other tracks. DCBETrackLock has an effect only when the device is defined as read-only.
22(16)	2	DCBERSV3	Reserved.
24(18)	8	DCBEBLKSI8	Block size in eight bytes.
28(1C)	4	DCBEBLKSI	BLKSIZE coded on DCBE macro or, if BLKSIZE=0 was coded, value set by OPEN. After OPEN, this field is valid only if OPEN set DCBESLBI on.
32(20)	8	DCBEXSIZ	Number of blocks in current data set. Set by system when DCBEGSIZ is set.
32(20)	4	DCBESIZO	High order word of DCBEXSIZ.
36(24)	4	DCBESIZE	Number of blocks in current data set. Set by system when DCBEGSIZ is set.
40(28)	4	DCBEEODA	Address of user provided end-of-data routine. May reside above or below the line. Used instead of DCBEODAD. Will be zero if no address is given.
44(2C)	4	DCBESYNA	Address of user provided SYNAD routine. May reside above or below the line. Used instead of DCBSYNAD. Will be zero if no address is given.
48(30)	4		Reserved.

Offset	Length or Bit Pattern	Field Name	Description
52(34)	2	DCBENMFL	Number of tape files written before synchronization (SYNC=nnn).
54(36)	1	DCBEMACC	MULTACC. Accumulation number multiplier.
55(37)	1	DCBEMSDN	MULTSDN. Multiplier of system determined NCP.
56(38)		DCBEMINL	Minimal length of DCBE in any release.
56(38)		DCBEEND	End of DCBE. This label is always after the last DCBE field.

Appendix C. Control characters

Each logical record (except with VSAM), in all record formats, can contain an optional control character. This control character controls stacker selection on a card punch or card read punch, or printer spacing and skipping. If a record containing an optional control character is directed to any other device, it is considered to be the first data byte, and it does not cause a control function to occur.

In format-F and format-U records, the optional control character must be in the first byte of the logical record. In format-V or format-D records, the optional control character must be in the fifth byte of the logical record, immediately following the record descriptor word of the record.

Two control character options are available: machine code and extended code defined by ANSI. Code the control character in the RECFM parameter of the DCB macro. If either option is specified in the data control block, you must include a control character in each record. Other spacing or stacker selection options also specified in the data control block are ignored.

Independently of control characters, each record can contain a table reference character. If each record contains a control character and a table reference character, the control character is first.

Machine code

The record format field in the data control block indicates that the machine code control character has been placed in each logical record. If the record is written, the appropriate byte must contain the command code bit configuration specifying both the write and the desired carriage or stacker select operation.

The machine code control characters for a printer are:

Print—Then Act	Action	Act Immediately without Printing
X'01'	Print only (no space, overprint)	
X'09'	Space 1 line	X'0B'
X'11'	Space 2 lines	X'13'
X'19'	Space 3 lines	X'1B'
X'5A'	The rest of the record is page mode data. This requires the use of the Print Services Facility (PSF) and a page mode printer such as an IBM 3800, IBM 3900, IBM 3820, or IBM 3827. The data may be sysout.	
X'89'	Skip to channel 1	X'8B'
X'91'	Skip to channel 2	X'93'
X'99'	Skip to channel 3	X'9B'
X'A1'	Skip to channel 4	X'A3'
X'A9'	Skip to channel 5	X'AB'
X'B1'	Skip to channel 6	X'B3'
X'B9'	Skip to channel 7	X'BB'
X'C1'	Skip to channel 8	X'C3'
X'C9'	Skip to channel 9	X'CB'

Print—Then Act	Action	Act Immediately without Printing
X'D1'	Skip to channel 10	X'D3'
X'D9'	Skip to channel 11	X'DB'
X'E1'	Skip to channel 12	X'E3'

The machine code control characters for a card punch device are as follows:

Control Code	Action
X'01'	Select stacker 1
X'41'	Select stacker 2
X'81'	Select stacker 3

Other command codes for specific devices are contained in IBM System Reference Library publications describing the control units or devices.

ISO/ANSI

In place of machine code, you can specify control characters defined by ISO/ANSI. These characters must be represented in EBCDIC code.

ISO/ANSI control characters for a printer are as follows:

Code	Action before Printing a Line
b	Space one line (blank code)
0	Space two lines
-	Space three lines
+	Suppress space (overprint existing line)
1	Skip to channel 1
2	Skip to channel 2
3	Skip to channel 3
4	Skip to channel 4
5	Skip to channel 5
6	Skip to channel 6
7	Skip to channel 7
8	Skip to channel 8
9	Skip to channel 9
A	Skip to channel 10
B	Skip to channel 11
C	Skip to channel 12
X'5A'	The rest of the record is page mode data. This requires the use of the Print Services Facility (PSF) and a page mode printer such as an IBM 3800, IBM 3900, IBM 3820, or IBM 3827. The data may be sysout.

ISO/ANSI control characters for a card punch device are as follows:

Code	Action after Punching a Card
V	Select punch pocket 1
W	Select punch pocket 2

These control characters include those defined by ANSI FORTRAN. If any other character is specified, it is interpreted as 'b' or V, depending on the device being used; no error indication is returned.

ISO/ANSI record control word and segment control word

Conversion of ISO/ANSI record control word

The ISO/ANSI record control word (RCW) is expressed in ASCII characters and is 4 bytes long (see Figure 8 on page 399). Note that the RCW is different from the code in the IBM record descriptor word (RDW). The RDW, expressed in binary, is the internal data management equivalent of the ISO/ANSI RCW.

For ISO/ANSI V4 tapes created specifying a CCSID other than ASCII, the RCW will continue to be expressed in ASCII.

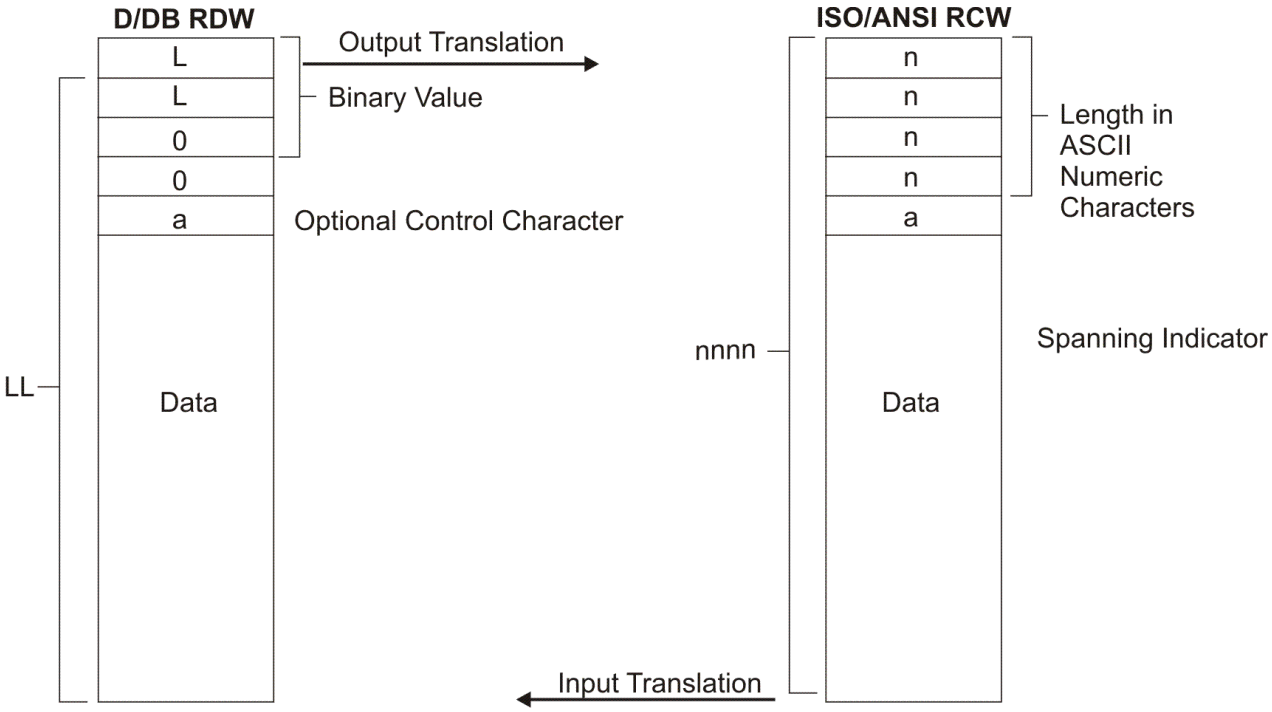
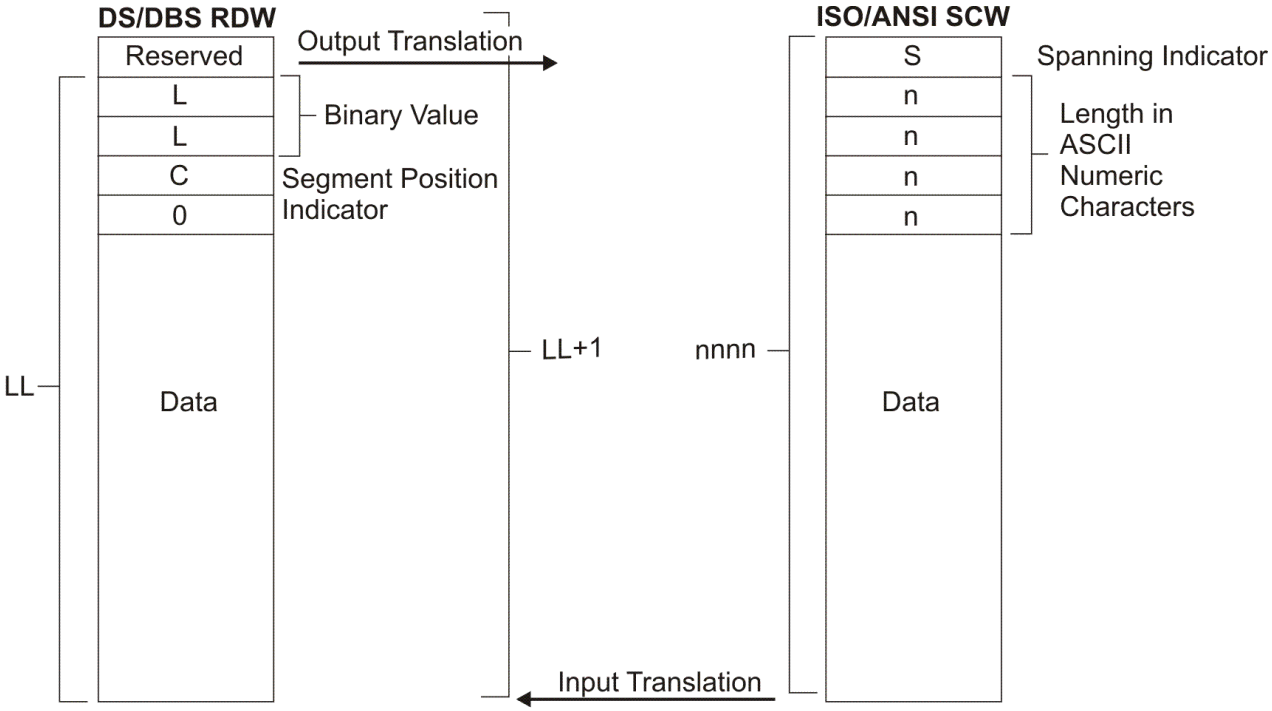


Figure 8. Conversion of ISO/ANSI Record Control Word to D/DB Record Descriptor Word

Conversion of ISO/ANSI segment control word

The ISO/ANSI segment control word (SCW) is expressed in ASCII characters and is 5 bytes in length (see Figure 9 on page 400). Note that the SCW is different from the code in the IBM segment descriptor word (SDW). The SDW is the internal data management equivalent of the ISO/ANSI SCW. Only 4 bytes are used by data management, but the user buffer area must accommodate an extra byte to allow for conversion from the ISO/ANSI SCW. The SDW is expressed in binary.



C values for SDW (2 low order bits)

00 = only segment of record
01 = first segment of record
11 = intermediate segment of record
10 = last segment of record

S values for SCW (ASCII characters)

0 = only segment of record
1 = first segment of record
2 = intermediate segment of record
3 = last segment of record

Figure 9. Conversion of ISO/ANSI Segment Control Word to DS/DBS Segment Descriptor Word

Appendix D. Index processing macros

This appendix is intended to help you to diagnose problems in the index of a VSAM data set.

You can use the macros documented here to examine the contents of the index of a key-sequenced data set, if your index is damaged or if pointers are lost. Two ways to access directly the index component of a key-sequenced data set or variable-length RRDS are:

- Open the data set as a cluster and use the GETIX and PUTIX macros to process a control interval.
- Open the index component as a data set and use the GET and PUT macros to process the index component as an entry-sequenced data set.

You should not attempt to duplicate or substitute the index processing done by VSAM during normal access to data records. It is best to let VSAM maintain all indexes.

GETIX—Retrieve an index record

The format of the GETIX macro is:

[*label*]	GETIX	RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the GETIX macro.

RPL=*address*
specifies the address of the request parameter list that defines this GETIX request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

The following RPL parameters and subparameters are required for GETIX:

OPTCD=(CNV
,DIR
,{NUP|UPD|NSP}
,{LOC|MVE})

GETIX can be issued either for update or not for update; OPTCD=NSP is interpreted as OPTCD=NUP.

With OPTCD=MVE, AREALEN must be at least index control interval size.

ARG=*address*
The search argument for GETIX is the RBA of a control interval.

To process the index of a key-sequenced data set with GETIX, you must open the cluster with:

```
ACB MACRF=(CNV,...)
```

PUTIX—Store an index record

The format of the PUTIX macro is:

[*label*]	PUTIX	RPL=*address*

label
specifies 1 to 8 characters that provide a symbolic address for the PUTIX macro.

RPL=*address*
specifies the address of the request parameter list that defines this PUTIX request. You may specify the address in register notation (using a register from 1 through 12, enclosed in parentheses) or specify it with an expression that generates a valid relocatable A-type address constant.

The following RPL parameters and subparameters are required for PUTIX:

OPTCD=(CNV
,DIR
,UPD
,MVE)

OPTCD=LOC is not allowed.

AREALEN
must be at least index control interval size.

The contents of a control interval must previously have been retrieved for update through GETIX.

To process the index of a key-sequenced data set with GETIX, you must open the cluster with:

```
ACB MACRF=(CNV,...)
```

Appendix E. Selecting logical record lengths and block sizes for specific devices

The following information provides a guide to coding the block size (BLKSIZE) and logical record length (LRECL) operands in the DCB macro. These values can be used to determine the maximum block size and logical record length for a given device, and to determine the optimum blocking factor when records are to be blocked.

Printers

Table 70 on page 403 shows the record length that can be specified for the various printers.

Table 70. Record length for printers. Sometimes two values are shown; except for the 3800, the larger of the two values requires that an optional feature be installed on the printer being used. If the optional control character is specified to control spacing and skipping, specify the record length as one greater than the actual data length (the control character is not part of the data record). Another factor that affects record length is the presence of table reference characters (TRCs). If you specify OPTCD=J, then increase the record length by one to show that each record has a TRC. Any combination of control characters and table reference characters is valid.

Printer	Record Length (Bytes)
1403 Printer	120 or 132
3203 Printer	132
3211 Printer	132 or 150
3525 Card Punch, Print Feature	64
3800 Printing Subsystem	136 bytes for 10 pitch 163 bytes for 12 pitch 204 bytes for 15 pitch
4245 Printer	132
4248 Printer	132 or 168
3262 Model 5 Printer	132
6262 Printer	133

Table 70. Record length for printers. Sometimes two values are shown; except for the 3800, the larger of the two values requires that an optional feature be installed on the printer being used. If the optional control character is specified to control spacing and skipping, specify the record length as one greater than the actual data length (the control character is not part of the data record). Another factor that affects record length is the presence of table reference characters (TRCs). If you specify OPTCD=J, then increase the record length by one to show that each record has a TRC. Any combination of control characters and table reference characters is valid. *(continued)*

Printer	Record Length (Bytes)
AFP1 Device **3825-001** **3827-001** **3828-001** **3835-001** **3900-001**	32,760 "1" on page 404

Note:

1. Advanced function printers (page printers) can place a byte anywhere on a page and are not limited to formatted print lines. Therefore, the printers can use the full 32,760 byte records that the various systems support.

   When printing formatted print lines, the length of the line varies depending on the size of the font and paper.

Card readers and card punches

Format F, V, or U records are accepted by readers and punchers, but the logical record length for a card reader or card punch is fixed at 80 bytes. If the optional control character is specified, the logical record length is 81 (the control character is not part of the data record). If card image mode is used, the buffer required to contain the data must be 160 bytes.

Magnetic tape units

The following table describes the maximum block size that is supported by BSAM and QSAM on magnetic tape devices (in bytes).

Tape	Without LBI	Optimum with LBI "1" on page 405	Maximum with LBI
3420 Magnetic Tape Units (7 track and 9 track)	32,760	32,760	32,760
3430 Magnetic Tape Units	32,760	32,760	32,760
3480 Magnetic Tape Subsystem (with or without compaction mode)	32,760	65,535	65,535
3490 and 3490E Magnetic Tape Subsystem (with or without compaction mode)	32,760	65,535	65,535

Tape	Without LBI	Optimum with LBI[“1” on page 405]	Maximum with LBI
3590	32,760	229,376 or 262,144	262,144

Note:

1. LBI stands for large block interface. You can request it by coding the BLKSIZE keyword on the DCBE macro or turning on the DCBEULBI bit before completion of the DCB OPEN exit routine. If the system accepts the request, OPEN turns on the DCBESLBI in the DCBE after the DCB OPEN exit routine. This bit affects several BSAM and QSAM interfaces. Refer to *z/OS DFSMS Using Data Sets*.

Direct access storage devices

Each record that is written on direct access storage devices requires some *device overhead*.

Use the TRKCALC macro to calculate the exact number of bytes required for each data block including the space that is required for device overhead. For more information on how to use the TRKCALC macro, see *z/OS DFSMSdfp Advanced Services*.

If the TRKCALC macro cannot be used and space calculations must be performed manually, refer to the appropriate Direct Access Storage Reference Summary.

The following tables help you estimate your space needs.

Table 71 on page 405 lists the physical characteristics of DASDs. Today, disk storage subsystems emulate the track capacity of an IBM 3380 or 3390 device while providing much larger capacity than the original 3380 and 3390 devices. For example, the IBM System Storage DS8000 series emulates emulates the IBM 3390. On an emulated disk or on a VM minidisk, the number of cylinders per volume is a configuration option. It might be less than or greater than the stated number. If so, the number of bytes per device differ accordingly. The IBM ESS Model 2105 supports up to 65520 cylinders and the IBM DS8000® supports up to 1,182,006 cylinders.

In a compressed format data set, PDSE z/OS UNIX file or dummy data set, the most efficient block size is unrelated to the track length. The most efficient block size would be up to 32760.

The maximum data length for a track multiplied by the number of tracks per cylinder produces the number of bytes available per cylinder for a device.

Similarly, the number of bytes per cylinder multiplied by the number of cylinders per volume produces the total number of bytes available for a device.

Table 71. DASD Physical Characteristics

Type	Most Efficient Block Size	Maximum Data Length/Track	Trk/Cyl	Bytes/Cyl Avail for User Records	Cyl/Vol
3390 emulation using IBM DS8000 device	27,998[1]	56,664	15	849,960	1 – 1,182,006
3380 emulation using IBM DS8000 device	23,476[1]	47,476	15	712,140	1 – 1,182,006
3390 Model 1	27,998[1]	56,664	15	849,960	1,113
3390 Model 3	27,998[1]	56,664	15	849,960	3,339
3390 Model 9	27,998[1]	56,664	15	849,960	10,017
9345 Model 1	22,928[1]	46,456	15	696,840	1,440
9345 Model 2	22,928[1]	46,456	15	696,840	2,156

Note:

1. Two-record format

Basic access methods track capacity

The BAM (basic access methods) is BSAM, BPAM, QSAM, and BDAM. EXCP is also considered a BAM. VSAM emulates some ISAM functions in a VSAM data set. VSAM is not described in this section.

You should allow the operating system to choose an optimal block size for new data sets unless you have a reason to set a particular value. The system considers the data set type and other characteristics when choosing a block size if the data set does not have an undefined record format (RECFM=U). Because the largest record that is supported by the access methods is a slightly less than 32 KB, the most efficient block size is not necessarily the maximum data length that can fit on the track.

For example, to maximize use of a 3390 track, 98.8% of the space available on a track can be used by writing two records of 27998 bytes each. The most efficient block size for the 3380 would be 23476 bytes. Two of these blocks would fit on a 3390 track. However, in some cases, you must consider data set type when determining the most efficient block size. For example, extended format data sets have a 32-byte suffix for each block and encrypted basic and large format data sets have an 8-byte prefix for each block. In these cases, the most efficient block size (BLKSIZE as seen by the user) must be adjusted to take into account the length of the suffix or prefix.

In a load module PDS, allow the binder to set the block size. The record format is U, and most records are shorter than the maximum.

With compressed format data sets and with PDSEs, the actual block size is not under your control. The block size as seen by the application program is the BLKSIZE value. It is simulated by the access method. In those cases, the most efficient block size is up to 32 KB.

Table 72 on page 406 shows how many bytes you can write on each track for a 3390 if each block is the same length and does not have a key. A key refers to the KEYLEN parameter on the DCB macro or DD statement.

If you write fixed-length records with lengths as shown in the Data Length Range column on the left side of the table, see the result in the last three columns. The Percent Space Used column shows what percentage of the track is occupied by user data if the track has the maximum length for the data length range. The Maximum Track Capacity column shows how many records fit on a track and the number of bytes on that track.

For example, the most efficient block size that BAM supports on a 3390 is 27998 but if your record format is fixed-blocked (FB), the block size must be a multiple of the logical record size (LRECL). Suppose that you have LRECL=80 and RECFM=FB, the most efficient block size is the largest value that is a multiple of 80 but no more than 27998. Using integer arithmetic, the calculation is 27998 ÷ 80 × 80 or 27920.

The capacity of a cylinder is 15 times the track capacity.

Table 72. 3390 track capacity without keys

Data Length Range		**Percent Space Used**	**Maximum Track Capacity**	
Min	**Max**	**Used**	**Records**	**Bytes**
27 999	56 664	100.0	1	56 664
18 453	27 998	98.8	2	55 996
13 683	18 452	97.7	3	55 356
10 797	13 682	96.6	4	54 728
8 907	10 796	95.3	5	53 980
7 549	8 906	94.3	6	53 436
6 519	7 548	93.2	7	52 836
5 727	6 518	92.0	8	52 144

Table 72. 3390 track capacity without keys (continued)

Data Length Range		Percent Space Used	Maximum Track Capacity	
5 065	5 726	90.9	9	51 534
4 567	5 064	89.4	10	50 640
4 137	4 566	88.6	11	50 226
3 769	4 136	87.6	12	49 632
3 441	3 768	86.4	13	48 984
3 175	3 440	85.0	14	48 160
2 943	3 174	84.0	15	47 610
2 711	2 942	83.1	16	47 072
2 547	2 710	81.3	17	46 070
2 377	2 546	80.9	18	45 828
2 213	2 376	79.7	19	45 144
2 083	2 212	78.1	20	44 240
1 947	2 082	77.2	21	43 722
1 851	1 946	75.6	22	42 812
1 749	1 850	75.1	23	42 550
1 647	1 748	74.0	24	41 952
1 551	1 646	72.6	25	41 150
1 483	1 550	71.1	26	40 300
1 387	1 482	70.6	27	40 014
1 319	1 386	68.5	28	38 808
1 251	1 318	67.5	29	38 222
1 183	1 250	66.2	30	37 500
1 155	1 182	64.7	31	36 642
1 087	1 154	65.2	32	36 928
1 019	1 086	63.2	33	35 838
985	1 018	61.1	34	34 612
951	984	60.8	35	34 440
889	950	60.4	36	34 200
855	888	58.0	37	32 856
821	854	57.3	38	32 452
787	820	56.4	39	31 980
753	786	55.5	40	31 440
719	752	54.4	41	30 832
691	718	53.2	42	30 156

Table 72. 3390 track capacity without keys (continued)

Data Length Range		Percent Space Used	Maximum Track Capacity	
657	690	52.4	43	29 670
623	656	50.9	44	28 864
589	622	49.4	45	27 990
555	588	47.7	46	27 048
521	554	46.9	48	26 592
487	520	45.0	49	25 480
459	486	42.9	50	24 300
425	458	42.0	52	23 816
391	424	40.4	54	22 896
357	390	37.9	55	21 450
323	356	35.8	57	20 292
289	322	33.5	59	18 998
255	288	31.0	61	17 568
227	254	28.7	64	16 256
193	226	26.3	66	14 916
159	192	23.4	69	13 248
125	158	20.1	72	11 376
91	124	16.4	75	9 300
57	90	12.4	78	7 020
23	56	8.1	82	4 592
1	22	3.3	86	1 892

VSAM usage of space for selected device types

The following tables show how much space VSAM uses for various DASDs. See Table 71 on page 405 for the physical characteristics of DASDs.

Use these charts to select control interval sizes that make the most efficient use of storage. Refer to “Control interval size for selected devices” on page 411 to determine if the control interval size is equal to the physical block size of the data component.

Remember that the physical block size for the data component may be different from the physical block size for the index component, even if the same control interval size is selected for both. Thus, while the physical block size for the index component is always equal to the control interval size, the physical block size for the data component may be smaller than the control interval size. Multiple physical blocks may compose one control interval for the data component.

For example, if a facility is using the 3390 DASD and a control interval size of 14336 is selected for both the index and data components, then the index component will have a physical block size of 14336 (one physical block per control interval) and the data will have a physical block size of 7196 (two physical blocks per control interval).

Note: If direct, sequential, or partitioned data sets, or PDSEs are used, all without keys, and the size of the block matches the size of one of the control intervals given in these tables, then the corresponding information in the data columns can be used.

If the exact block size is not listed, see the appropriate Direct Access Storage Reference Summary.

VSAM usage of 3380 DASD space

Table 73. VSAM Usage of 3380 DASD Space

	Block Size		Physical Block/ Track		% Track Used			Bytes/Track	
CI Size	Data	Index	Data	CI/CA	Index	Data	Index	Data	Index
512	512	512	46	46	690	49.61	49	23,552	23,552
1,024	1,024	1,024	31	31	465	66.86	66	31,744	31,744
1,536	1,536	1,536	23	23	345	74.41	74	35,328	35,328
2,048	2,048	2,048	18	18	270	77.65	77	36,864	36,864
2,560	2,560	2,560	15	15	225	80.88	80	38,400	38,400
3,072	3,072	3,072	13	13	195	84.12	84	39,936	39,936
3,584	3,584	3,584	11	11	165	83.04	83	39,424	39,424
4,096	4,096	4,096	10	10	150	86.28	86	40,960	40,960
4,608	4,608	4,608	9	9	135	87.35	87	41,472	41,472
5,120	5,120	5,120	8	8	120	86.28	86	40,960	40,960
5,632	5,632	5,632	7	7	105	83.04	83	39,424	39,424
6,144	6,144	6,144	7	7	105	90.59	90	43,008	43,008
6,656	6,656	6,656	6	6	90	84.12	84	39,936	39,936
7,168	7,168	7,168	6	6	90	90.59	90	43,008	43,008
7,680	7,680	7,680	5	5	75	80.88	80	38,400	38,400
8,192	8,192	8,192	5	5	75	86.28	86	40,960	40,960
10,240	10,240	10,240	4	4	60	86.28	86	40,960	40,960
12,288	6,144	12,288	7	3	52	90.59	77	43,008	36,864
14,336	14,336	14,336	3	3	45	90.59	90	43,008	43,008
16,384	8,192	16,384	5	2	37	86.28	69	40,960	32,768
18,432	6,144	18,432	7	2	35	90.59	77	43,008	36,864
20,480	20,480	20,480	2	2	30	86.28	86	40,960	40,960
22,528	22,528	22,528	2	2	30	94.90	94	45,056	45,056
24,576	6,144	24,576	7	1	26	90.59	51	43,008	24,576
26,624	6,656	26,624	6	1	22	84.12	56	39,936	26,624
28,672	14,336	28,672	3	1	22	90.59	60	43,008	28,672
30,720	6,144	30,720	7	1	21	90.59	64	43,008	30,720
32,768	8,192	32,768	5	1	18	86.28	69	40,960	32,768

VSAM usage of 3390 DASD space

Table 74. VSAM Usage of 3390 DASD Space

	Block Size		Physical Block/ Track			% Track Used		Bytes/Track	
CI Size	**Data**	**Index**	**Data**	**Index**	**CI/CA**	**Data**	**Index**	**Data**	**Index**
512	512	512	49	49	735	44.28	43	25,088	25,088
1,024	1,024	1,024	33	33	495	59.64	58	33,792	33,792
1,536	1,536	1,536	26	26	390	70.48	69	39,936	39,936
2,048	2,048	2,048	21	21	315	75.90	74	43,008	43,008
2,560	2,560	2,560	17	17	255	76.80	75	43,520	43,520
3,072	3,072	3,072	15	15	225	81.32	79	46,080	46,080
3,584	3,584	3,584	13	13	195	82.22	80	46,592	46,592
4,096	4,096	4,096	12	12	180	86.74	85	49,152	49,152
4,608	4,608	4,608	10	10	150	81.32	79	46,080	46,080
5,120	5,120	5,120	9	9	135	81.32	79	46,080	46,080
5,632	5,632	5,632	9	9	135	89.45	87	50,688	50,688
6,144	6,144	6,144	8	8	120	86.74	85	49,152	49,152
6,656	6,656	6,656	7	7	105	82.22	80	46,592	46,592
7,168	7,168	7,168	7	7	105	89.50	86	50,176	50,176
7,680	7,680	7,680	6	6	90	81.32	79	46,080	46,080
8,192	8,192	8,192	6	6	90	86.74	85	49,152	49,152
10,240	10,240	10,240	5	5	75	90.36	88	51,200	51,200
12,288	12,288	12,288	4	4	60	86.74	85	49,152	49,152
14,336	7,168	14,336	7	3	52	89.50	74	50,176	43,008
16,384	16,384	16,384	3	3	45	86.74	85	49,152	49,152
18,432	18,432	18,432	3	3	45	97.59	95	55,296	55,296
20,480	10,240	20,480	5	2	37	90.36	70	51,200	40,960
22,528	5,632	22,528	9	2	33	89.45	77	50,688	45,056
24,576	24,576	24,576	2	2	30	86.74	85	49,152	49,152
26,624	26,624	26,624	2	2	30	93.97	92	53,248	53,248
28,672	7,168	28,672	7	1	26	89.50	49	50,176	28,672
30,720	10,240	30,720	5	1	25	90.36	53	51,200	30,720
32,768	16,384	32,768	3	1	22	86.74	56	49,152	32,768

Note:

For extended-format VSAM data sets, each block has a 32-byte suffix which is added by the system. You will need to take this into account when you calculate disk space requirements. For more information, see Extended-Format VSAM Data Sets in *z/OS DFSMS Using Data Sets*.

VSAM usage of 9345 DASD space

Table 75. VSAM Usage of 9345 DASD Space

CI Size	Block Size		Physical Block/ Track		CI/CA	% Track Used		Bytes/Track	
	Data	Index	Data	Index		Data	Index	Data	Index
512	512	512	41	41	615	45.19	43	20,992	20,992
1,024	1,024	1,024	28	28	420	61.72	59	28,672	28,672
1,536	1,536	1,536	21	21	315	69.43	66	32,256	32,256
2,048	2,048	2,048	17	17	255	74.94	72	34,816	34,816
2,560	2,560	2,560	14	14	210	77.15	74	35,840	35,840
3,072	3,072	3,072	12	12	180	79.35	76	36,864	36,864
3,584	3,584	3,584	11	11	165	84.86	81	39,424	39,424
4,096	4,096	4,096	10	10	150	88.17	84	40,960	40,960
4,608	4,608	4,608	8	8	120	79.35	76	36,864	36,864
5,120	5,120	5,120	8	8	120	88.17	84	40,960	40,960
5,632	5,632	5,632	7	7	105	84.86	81	39,424	39,424
6,144	6,144	6,144	6	6	90	79.35	76	36,864	36,864
6,656	6,656	6,656	6	6	90	85.96	82	39,936	39,936
7,168	7,168	7,168	6	6	90	92.58	89	43,008	43,008
7,680	7,680	7,680	5	5	75	82.66	79	38,400	38,400
8,192	8,192	8,192	5	5	75	88.17	84	40,960	40,960
10,240	10,240	10,240	4	4	60	88.17	84	40,960	40,960
12,288	4,096	12,288	10	3	50	88.17	76	40,960	36,864
14,336	14,336	14,336	3	3	45	92.58	89	43,008	43,008
16,384	8,192	16,384	5	2	37	88.17	67	40,960	32,768
18,432	18,432	18,432	2	2	30	79.35	76	36,864	36,864
20,480	20,480	20,480	2	2	30	88.17	84	40,960	40,960
22,528	22,528	22,528	2	2	30	96.99	93	45,056	45,056
24,576	8,192	24,576	5	1	25	88.17	50	40,960	24,576
26,624	6,656	26,624	6	1	22	85.96	55	39,936	26,624
28,672	14,336	28,672	3	1	22	92.58	59	43,008	28,672
30,720	10,240	30,720	4	1	20	88.17	63	40,960	30,720
32,768	8,192	32,768	5	1	18	88.17	67	40,960	32,768

Control interval size for selected devices

For each DASD, Table 76 on page 412 identifies the control interval sizes that exactly fit one physical block for the data component. An X in the column indicates that for that device and the control interval size listed, the size of the control interval is equal to the physical block size of the data component.

If the chart does not show an X for a control interval size for a device, the control interval holds more than one block from the data component, and the control interval size listed is a multiple of the data component physical block size.

The control interval size is always equal to the index component physical block size.

Table 76. Control Interval Size

CI Size	Device		
	3380"1" on page 412	**3390**"2" on page 412	**9345**"3" on page 412
512	X"4" on page 412	X	X
1,024	X	X	X
1,536	X	X	X
2,048	X	X	X
2,560	X	X	X
3,072	X	X	X
3,584	X	X	X
4,096	X	X	X
4,608	X	X	X
5,120	X	X	X
5,632	X	X	X
6,144	X	X	X
6,656	X	X	X
7,168	X	X	X
7,680	X	X	X
8,192	X	X	X
10,240	X	X	X
12,288		X	
14,336	X		X
16,384		X	
18,432		X	X
20,480	X		X
22,528	X		X
24,576		X	
26,624		X	

Notes:

1. 3380, all models
2. 3390, all models
3. 9345, all models
4. X = VSAM-selected physical record size (equal to CI-size)

For more information on allocating space for a data set, see *z/OS DFSMS Using Data Sets*.

Appendix F. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com®/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS™, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming interface information

This book is intended to help you to use VSAM and non-VSAM macro instructions.

This publication documents intended programming interfaces that allow the customer to write programs to obtain services of DFSMS.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Glossary

This glossary defines technical terms and abbreviations used in DFSMS documentation. If you do not find the term you are looking for, refer to the index of the appropriate DFSMS manual.

This glossary includes terms and definitions from:

- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published part of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.
- The *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

The following cross-reference is used in this glossary:

See:
This refers the reader to (a) a related term, (b) a term that is the expanded form of an abbreviation or acronym, or (c) a synonym or more preferred term.

Numbers and Special Characters

3590B1*x*
An IBM TotalStorage Enterprise Tape Drive 3590 Model B1*x* that uses the 3590 High Performance Cartridge, writes in 128-track format, and can emulate the 3490 Magnetic Tape System.

3590E1*x*
An IBM TotalStorage Enterprise Tape Drive 3590 Model E1*xx* that uses the 3590 High Performance Cartridge, can read 128- or 256-track format tapes, and writes in 256-track format. This drive emulates either the IBM 3490 magnetic tape drive or the IBM TotalStorage Enterprise Tape Drive 3590 Model B1*x*.

3590H1*x*
An IBM TotalStorage Enterprise Tape Drive 3590 Model H1*xx* that uses the 3590 High Performance Cartridge, can read 128-, 256-, or 384-track format tapes, and writes in 384-track format. This drive emulates either the IBM 3490 magnetic tape drive or the IBM TotalStorage Enterprise Tape Drive 3590 Model B1*x* or Model E1*x*.

3592J1A
An IBM TotalStorage Enterprise Tape Drive 3592 that uses the 3592 Enterprise Tape Cartridge and writes in enterprise format 1 (EFMT1). This drive emulates either the IBM 3490 magnetic tape drive or the IBM TotalStorage Enterprise Tape Drive 3590 Model B1*x*.

A

ABEND
Abnormal end of task. End of a task, a job, or a subsystem because of an error condition that cannot be resolved by recovery facilities while the task is performed.

ABSTR
Absolute track (value of SPACE).

ACB
Access method control block (VSAM).

ACC
Accept erroneous block (value of EROPT).

access method control block (ACB)
A control block that links an application program to VSAM or VTAM programs.

access method services
A multifunction service program that manages VSAM and non-VSAM data sets, as well as catalogs. Access method services provides the following functions:

- defines and allocates space for data sets and catalogs
- converts indexed-sequential data sets to key-sequenced data sets
- modifies data set attributes in the catalog
- reorganizes data sets
- facilitates data portability among operating systems
- creates backup copies of data sets
- assists in making inaccessible data sets accessible
- lists the records of data sets and catalogs
- defines and builds alternate indexes

ACS
Automatic class selection.

addressed-direct access
In VSAM, the retrieval or storage of a data record identified by its relative byte address (RBA), independent of the record's location relative to the previously retrieved or stored record.

addressed-sequential access
In VSAM, the retrieval or storage of a data record in its entry sequence relative to the previously retrieved or stored record.

addressing mode (AMODE)
An attribute of an entry point in a program that identifies the addressing range in virtual storage which the module is capable of addressing. In 24-bit addressing mode, only 24-bit addresses can be used.

AIX
Alternate index.

AL
American National Standard labels.

alias
An alternative name for a catalog entry or for a member of a partitioned data set (PDS).

alias entry
An entry that relates an alias to the real entry name of a user catalog or non-VSAM data set.

allocation
(1) Generically, the entire process of obtaining a volume and unit of external storage, and setting aside space on that storage for a data set. (2) The process of connecting a program to a data set or devices.

alternate index (AIX)
A key-sequenced data set containing index entries organized by the alternate keys of its associated base data records. It provides an alternate means of locating records in the data component of a cluster on which the alternate index is based.

alternate key
One or more characters within a data record used to identify the data record or control its use. Unlike the prime key, the alternate key can identify more than one data record. It is used to build an alternate index or to locate one or more base data records via an alternate index.

AMODE
Addressing mode.

ANSI
American National Standards Institute

application
The use to which an access method is put or the end result that it serves; contrasted to the internal operation of the access method.

AUL
American National Standard user labels (value of LABEL).

B

BCD
Binary coded decimal.

BCDIC
Binary coded decimal interchange code.

BDAM
Basic direct access method.

BDW
Block descriptor word.

BFALN
Buffer alignment (DCB and parameter).

BFTEK
Buffer technique (DCB and parameter).

binder
The DFSMS/MVS program that processes the output of language translators and compilers into an executable program (load module or program object). It replaces the linkage editor and batch loader in MVS

BLKSIZE
Block size (DCB, DCBE and DD parameter).

blocking
The process of combining two or more records into one block.

block size
The number of records, words, or characters in a block; usually specified in bytes.

BLT
Block locator token.

BPAM
Basic partitioned access method.

BSAM
Basic sequential access method.

BSM
Backspace to tape mark.

BSP
Backspace one block (BSAM macro).

BSR
Backspace over a specified number of blocks (CNTRL parameter).

BUFC
Buffer control block (VSAM).

BUFCB
Buffer pool control block (DCB parameter).

BUFL
Buffer length (DCB and parameter).

BUFNO
Buffer number (DCB and parameter).

BUFOFF
Buffer offset.

C

CAT
Character arrangement table.

catalog
A data set that contains extensive information required to locate other data sets, to allocate and deallocate storage space, to verify the access authority of a program or operator, and to accumulate data set usage statistics. See *master catalog*.

CBIC
Control blocks in common.

CCHHR
Cylinder/head record address.

CCID
Coded character set identifier.

CF
Coupling facility.

CI
Control interval.

CIDF
Control interval definition field.

class
See *SMS class*.

cluster
In VSAM, a named structure consisting of a group of related components. For example, when the data is key-sequenced, the cluster contains both the data and index components; for data that is entry-sequenced, the cluster contains only a data component.

component
A named, cataloged collection of stored records. A component, the lowest member of the hierarchy of data structures that can be cataloged, contains no named subsets.

compress
To reduce the amount of storage required for a given data set by having the system replace identical words or phrases with a shorter token associated with the word or phrase.

To reclaim the unused and unavailable space in a partitioned data set that results from deleting or modifying members by moving all unused space to the end of the data set.

compressed format data set
A type of extended format data set created in a data format which supports record level compression.

configuration
The arrangement of a computer system as defined by the characteristics of its functional units.

See *SMS configuration*.

CONTIG
Contiguous space allocation (value of SPACE).

control blocks in common (CBIC)
A facility that allows a user to open a VSAM data set so the VSAM control blocks are placed in the common service area (CSA) of the MVS operating system. This provides the capability for multiple memory accesses to a single VSAM control structure for the same VSAM data set.

control interval (CI)
A fixed-length area of auxiliary storage space in which VSAM stores records. It is the unit of information (an integer multiple of block size) transmitted to or from auxiliary storage by VSAM.

control interval definition field (CIDF)
In VSAM, the 4 bytes at the end of a control interval that contain the displacement from the beginning of the control interval to the start of the free space and the length of the free space. If the length is 0, the displacement is to the beginning of the control information.

control program
A routine, usually part of an operating system, that aids in controlling the operations and managing the resources of a computer system.

control unit
A hardware device that controls the reading, writing, or displaying of data at one or more input/output devices. See also *storage control.*

cross memory
A synchronous method of communication between address spaces.

CSECT
Control section.

CVOL
Control volume.

CVT
Communication vector table.

CYLOFL
Number of tracks for cylinder overflow records (DCB parameter).

D

DA
Direct access (value of DEVD or DSORG).

DADSM
Direct access device space management.

DASD
Direct access storage device.

data class
A collection of allocation and space attributes, defined by the storage administrator, that are used to create a data set.

data extent block (DEB)
A control block that describes the physical attributes of the data set.

Data Facility Product (DFP)
An IBM licensed program used to manage programs, devices, and data in a z/OS environment.

data record
A collection of items of information from the standpoint of its use in an application, as a user supplies it to the system storage. Contrast with *index record*

data security
Prevention of access to or use of data or programs without authorization. As used in this publication, the safety of data from unauthorized use, theft, or purposeful destruction.

data set control block (DSCB)
A control block in the VTOC that describes data set characteristics.

data synchronization
The process by which the system ensures that data previously given to the system via WRITE, CHECK, PUT, and PUTX macros is written to some form of non-volatile storage.

DAU
Direct access unmovable data set (value of DSORG).

DCB
Data control block.

DCBE
Data control block extension.

DD
Data definition.

DEB
Data extent block.

DECB
Data event control block.

DEN
Magnetic tape density (DCB parameter).

DEVD
Device-dependent (DCB parameter).

device number
The reference number assigned to any external device.

DFSMSdfp
A DFSMS functional component or basic element of z/OS that provides functions for storage management, data management, program management, device management, and distributed data access.

dictionary
A table that associates words, phrases, or data patterns to shorter tokens. The tokens replace the associated words, phrases, or data patterns when a data set is compressed.

direct access
The retrieval or storage of data by a reference to its location in a data set rather than relative to the previously retrieved or stored data. See also *addressed-direct access*.

direct access device space management (DADSM)
A DFSMSdfp component used to control space allocation and deallocation on DASD.

direct data set
A data set whose records are in random order on a direct access volume. Each record is stored or retrieved according to its actual address or its address according to the beginning of the data set. Normally accessed via BDAM.

directly-allocated printer
A printer that is allocated to the application program.

DISP
Data set disposition (parameter of DD statement and SETPRT macro).

DPI
Data protection image.

DSCB
Data set control block.

DSECT
Dummy section (also called dummy control section).

DSORG
Data set organization (DCB parameter).

dynamic buffering
A user-specified option that requests that the system handle acquisition, assignment, and release of buffers.

E

ECB
Event control block.

entry-sequenced data set
A data set whose records are loaded without respect to their contents, and whose RBAs cannot change. Records are retrieved and stored by addressed access, and new records are added at the end of the data set.

EOD
End-of-data.

EODAD
End-of-data-set exit routine address (DCB, DCBE, and EXLST parameter).

EOKR
End-of-key range.

EOV
End-of-volume.

EROPT
Error options (DCB parameter).

ERP
Error recovery procedure.

ESA
Enterprise Systems Architecture.

ESA/370
Enterprise Systems Architecture, a hardware architecture unique to the IBM 3090™ Enhanced model processors and the 4381 Model Groups 91E and 92E. It reduces the effort required for managing data sets, removes certain MVS/XA constraints that limit applications, extends addressability for system, subsystem, and application functions, and helps exploit the full capabilities of SMS.

ESDS
Entry-sequenced data set (VSAM).

ESETL
End sequential retrieval (QISAM macro).

ESPIE
Extended specify program interruption exits.

ESTAE
Extended specify task abnormal exit.

exclusive control
Preventing multiple WRITE-add BDAM requests from updating the same dummy record or writing over the same available space on a track. When specified by the user, exclusive control requests that the system prevent the data block about to be read from being modified by other requests; it is specified in a READ macro and released in a WRITE or RELEX macro. When a WRITE-add request is about to be processed, the system automatically gets exclusive control of either the data set or the track.

EXCP
Execute channel program.

EXLST
Exit list (DCB and ACB parameter).

extended format
The format of a data set that has a data set name type (DSNTYPE) of EXTENDED. The data set is structured logically the same as a data set that is not in extended format but the physical format is different. Data sets in extended format can be striped or compressed. Data in an extended format VSAM KSDS can be compressed. See also *striped data set, compressed format*.

extent
A continuous space on a DASD volume occupied by a data set or portion of a data set.

F

FCB
Forms control buffer.

FEOV
Force end-of-volume (BSAM, QSAM macro).

field
In a record or control block, a specified area used for a particular category of data or control information.

FIPS
Federal Information Processing Standard.

format-D
ASCII variable-length records.

format-DB
ASCII variable-length, blocked records.

format-DBS
ASCII variable-length, blocked spanned records.

format-DS
ASCII variable-length, spanned records.

format-F
Fixed-length records.

format-FB
Fixed-length, blocked records.

format-FBS
Fixed-length, blocked, standard records.

format-FS
Fixed-length, standard records.

format-U
Undefined-length records.

format-V
Variable-length records.

format-VB
Variable-length, blocked records.

format-VBS
Variable-length, blocked, spanned records.

format-VS
Variable-length, spanned records.

free space
Space reserved within the control intervals of a key-sequenced data set for inserting new records into the data set in key sequence or for lengthening records already there; also, whole control intervals reserved in a control area for the same purpose.

FSM
Forward space to tape mark (CNTRL parameter).

FSR
Forward space over a specified number of blocks or records (CNTRL parameter).

G

GCR
Group coded recording.

GDGNT
Generation data group name table.

GEN
Generic key.

gigabyte
1 073 741 824 bytes.

GL
GET macro, locate mode (value of MACRF).

GM
GET macro, move mode (value of MACRF).

GSR
Global shared resources.

H

header label
An internal label, immediately preceding the first record of a file, that identifies the file and contains data used in file control.

The label or data set label that precedes the data records on a unit of recording medium.

HFS
see *hierarchical file system*

hierarchical file system (HFS) data set
A data set that contains a POSIX-compliant file system, which is a collection of files and directories organized in a hierarchical structure, that can be accessed using z/OS UNIX System Services. See also *file system.*

Hiperspace
A high performance virtual storage space of up to two gigabytes. Unlike an address space, a Hiperspace contains only user data and does not contain system control blocks or common areas; code does not execute in a Hiperspace. Unlike a data space, data in a Hiperspace cannot be referenced directly; data must be moved to an address space in blocks of 4KB before they can be processed. The 4K blocks can be backed by expanded storage or auxiliary storage, but never by virtual storage. The Hiperspace used by VSAM is only backed by expanded storage. See also *Hiperspace buffer*.

Hiperspace buffer
A 4K-byte-multiple buffer which facilitates the moving of data between a Hiperspace and an address space. VSAM Hiperspace buffers are only backed by expanded storage.

I

IDRC
Improved Data Recording Capability.

indexed VTOC
A volume table of contents with an index that contains a list of data set names and free space information, which allows data sets to be located more efficiently.

index record
In VSAM, a collection of index entries retrieved and stored as a group.

INOUT
Input then output (OPEN parameter).

IRF
Interrupt recognition flag.

IS
Indexed sequential (value of DSORG).

ISO
International Organization for Standardization

ISU
Indexed sequential unmovable (value of DSORG).

J

JFCB
Job file control block.

JFCBE
Job file control block extension.

K

KEYLEN
Key length (DCB and RPL parameter).

key-sequenced data set (KSDS)
A VSAM data set whose records are loaded in ascending key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in key sequence because of free space allocated in the data set. Relative byte addresses of records can change because of control interval or control area splits.

keyed sequential access
In VSAM, the retrieval or storage of a data record in its key or relative-record sequence, relative to the previously retrieved or stored record as defined by the sequence set of an index.

kilobyte
1024 bytes.

KSDS
Key-sequenced data set (VSAM).

L

LBI
Large block interface.

LDS
Linear data set (VSAM).

library
Synonym for partitioned data set. See *partitioned data set*.

linear data set (LDS)
A VSAM data set that contains data but no control information. A linear data set can be accessed as a byte-addressable string in virtual storage.

load module
The output of the linkage editor; a program in a format ready to load into virtual storage for execution. Contrast with program object.

locate mode
A transmittal mode in which a pointer to a record is provided instead of moving the record. Contrast with *move mode*.

LRECL
Logical record length (DCB parameter).

LRI
Logical record interface.

LSR
Local shared resources.

M

M
Mega.

MACRF
Macro form (DCB and ACB parameter).

management class
A collection of management attributes, defined by the storage administrator, used to control the release of allocated but unused space; to control the retention, migration, and backup of data sets; to control the retention and backup of aggregate groups, and to control the retention, backup, and class transition of objects.

MBBCCHHR
Module#, bin#, cylinder#, head#, record#.

member
A partition of a partitioned data set or PDSE.

MOD
Modify data set (value of DISP).

move mode
A transmittal mode in which the record to be processed is moved into a user work area.

MSHI
Virtual storage for highest-level index (DCB parameter).

MSWA
Virtual storage for work area (DCB parameter).

N

NCP
Number of channel programs (DCB parameter).

NFS
Network File System

non-VSAM data set
A data set allocated and accessed using one of the following methods: BDAM, BPAM, BISAM, BSAM, QSAM, QISAM.

NRZI
Nonreturn-to-zero-inverted.

NSL
Nonstandard label (value of LABEL).

NTM
Number of tracks in cylinder index for each entry in lowest level of master index (DCB parameter).

NUP
No update.

O

object
A named byte stream having no specific format or record orientation.

OMR
Optical mark read.

operand
Information entered with a command name to define the data on which a command operates and to control the execution of the command.

OPTCD
Optional services code (DCB and RPL parameter).

optimum block size
For non-VSAM data sets, optimum block size represents the block size that would result in the smallest amount of space utilization on a device, taking into consideration record length and device characteristics.

OUTIN
Output then input (OPEN parameter).

P

partitioned data set (PDS)
A data set on direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

partitioned data set extended (PDSE)
A data set that contains an indexed directory and members that are similar to the directory and members of partitioned data sets. A PDSE can be used instead of a partitioned data set.

path
A named, logical entity composed of one or more clusters (an alternate index and its base cluster, for example).

PDAB
Parallel data access block.

PDF
Problem determination function.

PDS
Partitioned data set.

PDS directory
A set of records in a partitioned data set (PDS) used to relate member names to their locations on a DASD volume.

PDSE
Partitioned data set extended.

PE
Phase encoding (tape recording mode).

PL
PUT macro, locate mode (value of MACRF).

PM
PUT macro, move mode (value of MACRF).

PMAR
Program management attribute record.

PO
Partitioned organization (value of DSORG).

pointer
An address or other indication of location. For example, an RBA is a pointer that gives the relative location of a data record or a control interval in the data set to which it belongs.

POU
Partitioned organization unmovable (value of DSORG).

primary space allocation
Amount of space requested by a user for a data set when it is created. Contrast with *secondary space allocation*.

prime key
One or more characters within a data record used to identify the data record or control its use. A prime key must be unique.

program library
A type of PDSE which contains program objects only. A PDSE from which programs are loaded into memory for execution by the operating system.

program object
All or part of a computer program in a form suitable for loading into virtual storage for execution. Program objects are stored in PDSE program libraries and have fewer restrictions than load modules. Program objects are produced by the binder.

PRTSP
Printer line spacing (DCB parameter).

PS
Physical sequential (value of DSORG).

PSF
Print Services Facility.

PSU
Physical sequential unmovable (value of DSORG).

Q

QSAM
Queued sequential access method.

QISAM
Queued indexed sequential access method.

R

R0
Record zero.

RACF
Resource Access Control Facility.

random access
See *direct access*.

RBA
Relative byte address.

RCW
Record control word.

RDBACK
Read backward (OPEN parameter).

RDF
Record definition field.

RDW
Record descriptor word.

RECFM
Record format (DCB parameter).

record definition field (RDF)
A field stored as part of a stored record segment; it contains the control information required to manage stored record segments within a control interval.

record level sharing
See *VSAM Record Level Sharing (VSAM RLS)*.

RECORG
Record organization.

register
An internal computer component capable of storing a specified amount of data and accepting or transferring this data rapidly.

relative byte address (RBA)
The displacement of a data record or a control interval from the beginning of the data set to which it belongs; independent of the manner in which the data set is stored.

relative record data set (RRDS)
A type of VSAM data set whose records have fixed or variable lengths, and are accessed by relative record number.

residence mode (RMODE)
The attribute of a load module that identifies where in virtual storage the program will reside (above or below 16 megabytes).

reusable data set
A VSAM data set that can be reused as a work file, regardless of its old contents. It must not be a base cluster of an alternate index.

RKP
Relative key position (DCB parameter).

RLS
See *VSAM Record Level Sharing (VSAM RLS)*.

RLSE
Release unused space (DD statement).

RMODE
Residence mode.

RPL
Request parameter list.

RRDS
Relative record data set (VSAM).

S

scheduling
The ability to request that a task set should be started at a particular interval or on occurrence of a specified program interrupt.

SCW
Segment control word.

SDW
Segment descriptor word.

secondary space allocation
Amount of additional space requested by the user for a data set when primary space is full. Contrast with *primary space allocation*.

security
See *data security*.

sequence checking
The process of verifying the order of a set of records relative to some field's collating sequence.

sequential access
The retrieval or storage of a data record in: its entry sequence, its key sequence, or its relative record number sequence, relative to the previously retrieved or stored record. See also *addressed-sequential access* and *keyed-sequential access*.

sequential data set
A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Normally accessed with BSAM or QSAM. Contrast with *direct data set* and *partitioned data set*.

SER
Volume serial number (value of VOLUME).

service request block (SRB)
A system control block used for dispatching tasks.

SETL
Set lower limit of sequential retrieval (QISAM macro).

SF
Sequential forward (parameter of READ or WRITE).

shared resources
A set of functions that permit the sharing of a pool of I/O-related control blocks, channel programs, and buffers among several VSAM data sets open at the same time.

SK
Skip to a printer channel (CNTRL parameter).

skip-sequential access
Keyed-sequential retrieval or storage of records here and there throughout a data set, skipping automatically to the desired record or collating position for insertion: VSAM scans the sequence set to find a record or a collating position. Valid for processing in ascending sequences only.

SKP
Skip erroneous block (value of EROPT).

SL
IBM standard labels (value of LABEL).

SLI
Suppress length indication bit.

slot
For a fixed-length relative record data set, the data area addressed by a relative record number which may contain a record or be empty.

SMF
System management facilities.

SMS
Storage Management Subsystem or system-managed storage.

SMS class
A list of attributes that SMS applies to data sets having similar allocation (data class), performance (storage class), or backup and retention (management class) needs.

SMS configuration
A configuration base, Storage Management Subsystem class, group, library, and drive definitions, and ACS routines that the Storage Management Subsystem uses to manage storage. See also *configuration, base configuration, source control data set* .

SMSI
Size of main-storage area for highest-level index (DCB parameter).

SMS-managed data set
A data set that has been assigned a storage class.

SMSW
Size of main-storage work area (DCB parameter).

SP
Space lines on a printer (CNTRL parameter).

spanned record
For VSAM, a logical record whose length exceeds control interval length, and as a result, crosses, or spans one or more control interval boundaries within a single control area. For non-VSAM, a spanned record that occupies part or all of more than one block.

SRB
Service request block.

SS
Select stacker on card reader (CNTRL parameter).

storage class
A collection of storage attributes that identify performance goals and availability requirements, defined by the storage administrator, used to select a device that can meet those goals and requirements.

storage control
The component in a storage subsystem that handles interaction between processor channel and storage devices, runs channel commands, and controls storage devices.

storage group
A collection of storage volumes and attributes, defined by the storage administrator. The collections can be a group of DASD volumes or tape volumes, or a group of DASD, optical, or tape volumes treated as a single object storage hierarchy. See also *VIO storage group, pool storage group, tape storage group, object storage group, object backup storage group, dummy storage group*.

Storage Management Subsystem (SMS)
A DFSMS facility used to automate and centralize the management of storage. Using SMS, a storage administrator describes data allocation characteristics, performance and availability goals, backup and retention requirements, and storage requirements to the system through data class, storage class, management class, storage group, and ACS routine definitions.

stripe
In DFSMS, the portion of a striped data set, such as an extended format data set, that resides on one volume. The records in that portion are not always logically consecutive. The system distributes records among the stripes such that the volumes can be read from or written to simultaneously to gain better performance. Whether it is striped is not apparent to the application program.

striping
A software implementation of a disk array that distributes a data set across multiple volumes to improve performance.

SUL
IBM standard and user labels (value of LABEL).

SWA
Scheduler work area.

SYNAD
Synchronous error routine address (DCB and DCBE parameter).

SYSIN
System input stream.

SYSOUT
System output stream.

system-managed storage
Storage managed by the Storage Management Subsystem. SMS attempts to deliver required services for availability, performance, and space to applications. See also *system-managed storage environment*.

DFSMS environment
An environment that helps automate and centralize the management of storage. This is achieved through a combination of hardware, software, and policies. In the DFSMS environment for z/OS, this function is provided by DFSMS, DFSORT, and RACF. See also *system-managed storage*.

system management facilities (SMF)
A component of MVS/ESA SP that collects input/output (I/O) statistics, provided at the data set and storage class levels, which helps you monitor the performance of the direct access storage subsystem.

T

T
Track overflow option (value of RECFM); user-totaling (value of OPTCD).

TIOT
Task I/O table.

transaction ID (TRANSID)
A number associated with each of several request parameter lists that define requests belonging to the same data transaction.

TRC
Table reference character.

TRTCH
Tape recording technique (DCB parameter).

TTR
Track record address.

U

UCB
Unit control block.

UCS
Universal character set.

unit address
The last two hexadecimal digits of a device address. This identifies the storage control and DAS string, controller, and device to the channel subsystem.

unit of recovery
A set of changes on one node that is committed or backed out as part of an ACID transaction.

A UR is implicitly started the first time a resource manager touches a protected resource on a node. A UR ends when the two-phase commit process for the ACID transaction changing it completes.

universal character set (UCS)
A printer feature that permits the use of a variety of character arrays. Character sets used for these printers are called UCS images.

UPD
Update.

update number
For a VSAM spanned record, a binary number in the second RDF of a record segment that indicates how many times the segments of a spanned record should be equal. An inequality indicates a possible error.

user buffering
The use of a work area in the processing program's address space for an I/O buffer; VSAM transmits the contents of a control interval between the work area and direct access storage without intermediary buffering.

z/OS UNIX file.
A collection of information treated as a unit. Examples of files used in z/OS UNIX System Services (z/OS UNIX) are HFS, ZFS, NFS, and TFS.

V

virtual storage access method (VSAM)
An access method for direct or sequential processing of fixed and variable-length records on direct access storage devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry sequence), or by relative record number.

VRRDS
Variable-length relative record data set.

VSAM
Virtual storage access method.

VSAM record-level sharing (VSAM RLS)
An extension to VSAM that provides direct record-level sharing of VSAM data sets from multiple address spaces across multiple systems. Record-level sharing uses the z/OS Coupling Facility to provide cross-system locking, local buffer invalidation, and cross-system data caching.

VSAM volume data set (VVDS)
A data set that describes the characteristics of VSAM and system-managed data sets residing on a given DASD volume; part of a catalog. See also *basic catalog structure*.

VSE
Virtual Storage Extended.

VSRT
VSAM shared resource table.

VTOC
Volume table of contents.

VVDS
VSAM volume data set.

W

WCGM
Writable character generation module.

X

XLRI
Extended logical record interface.

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Product Number: 5650-ZOS

SC23-6852-50