

GC28-0631-1

Systems

**OS/VS Data Management for
System Programmers**

**VS1 Release 1
VS2 Release 1**

IBM

Second Edition (September 1972)

This edition replaces the previous edition (numbered GC28-0631-0) and makes that edition obsolete.

This edition applies both to Release 1 of OS/VS1 and to Release 1 of OS/VS2, and to all subsequent releases of either system unless otherwise indicated in new editions or technical newsletters.

Significant technical and editorial changes made in this edition are outlined in "Summary of Major Changes" following the list of figures. This summary is followed by a brief discussion of differences between OS/VS Data Management for System Programmers and OS/MFT and OS/MVT Data Management for System Programmers. Each technical change is marked by a vertical line to the left of the change.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using the publication, consult the latest *IBM SRL Newsletter*, GN20-0360, that amends *IBM System/360 and System/370 Bibliography*, GA22-6822, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of this publication. If the forms have been removed, comments may be addressed to IBM Corporation, Programming Publications, Department D78, Monterey and Cottle Roads, San Jose, California 95114. All comments become the property of IBM.

© Copyright International Business Machines Corporation 1972

PREFACE

This publication provides information on how to modify and extend the data management capabilities of the OS/VS system control program; the intended audience is system programmers.

Some topics included are:

- Maintaining the System Catalog
- Maintaining the Volume Table of Contents
- Executing Your Own Channel Programs
- Using XDAP to Read and Write Data Sets on Direct Access Devices
- Password Protecting Your Data Sets

The OS/VS system control program provides simpler ways (for example, job control language, utility programs, access method routines) to do each of these things. The information presented in this book (consisting of macro specifications and how-to information) is intended to provide greater flexibility of implementation methods.

Other topics presented are:

- Using System Macro Instructions to Refer to, Validate, and Modify System Control Blocks
- Adding a UCS Image or FCB Image to the System Image Library.

This book makes reference to the DEB validity checking (DEBCHK) macro instruction and the authorized program facility (APF). This information applies only to installations using VS2; for installations using VS1, it is provided for planning purposes only.

Prerequisite Reading

Readers are expected to understand how to:

- Code programs in assembler language as described in *OS/VS and DOS/VS Assembler Language*, GC33-4010.
- Use the standard linkage conventions as described in *OS/VS Supervisor Services and Macro Instructions*, GC27-6979.
- Maintain the catalog and VTOC as described in *OS/VS JCL Services*, GC28-0617, *OS/VS Utilities*, GC35-0005, and *OS/VS Data Management Services Guide*, GC26-3783.
- Use the access methods to do input/output using the data management macros as described in *OS/VS Data Management Services Guide*, GC26-3783, and *OS/VS Data Management Macro Instructions*, GC26-3794.
- Protect data sets as described under “IEHPROGM” in *OS/VS Utilities*, GC35-0005.

More specific prerequisite reading is listed at the beginning of each chapter, as it relates to the particular topic.

Related Reading

All of the chapters of this publication make reference to:

- *OS/VS1 System Data Areas*, SY28-0605
- *OS/VS2 System Data Areas*, SY28-0606

These books present the detailed descriptions of system control blocks and common work areas.

More specific related reading is listed at the beginning of each chapter, as it relates to the topic under discussion.

How to Use This Book

You can use the macro specifications, coding examples, and how-to information in the chapter about catalog maintenance to code your own routines to add, delete, and update entries in the system catalog. This section contains data area layouts for and descriptions of the fields of all the control blocks that can appear in the system catalog.

If you want to read a data set control block, rename a data set, or delete a data set using the system macros, the chapter on maintaining the volume table of contents (VTOC) provides macro specifications, coding examples, and how-to information.

If you want to code your own channel programs to modify the control program or to provide support for unsupported I/O devices, the chapter on using EXCP provides detailed descriptions of which control blocks you must provide and the functions you must perform.

Macro specifications and how-to information are provided for using the XDAP macro instruction to read from and write to direct-access devices without using the access method routines (SAM, ISAM, or BDAM).

If you want to implement data set protection for your facility, the chapter on data set protection:

1. Tells how to build a PASSWORD data set.
2. Describes how the system control program responds to job control language and IEHPROGM utility statements in maintaining the PASSWORD data set.
3. Tells you how to use the PROTECT macro instruction to maintain (add records to, delete records from, changes records in) and read the PASSWORD data set.

The chapter on system macro instructions provides how-to information and macro specifications for:

1. Using system mapping macros to allow you to access system control blocks and work areas using symbolic names.
2. Examining device-type information in unit control blocks (UCBs).
3. Modifying a job file control block (JFCB) before opening a data set.
4. Removing queued requests and restoring requests to queues.
5. Protecting your data sets by verifying data extent blocks.

You can use the coding examples and how-to information in the last chapter to help you add a universal character set (UCS) image or a forms control buffer (FCB) image to the system image library (SYS1.IMAGELIB).

CONTENTS

<i>Page</i>	
iii	Preface
iii	Prerequisite Reading
iv	Related Reading
iv	How to Use This Book
ix	Figures
xi	Summary of Major Changes
xiii	How OS/VS Differs From OS/MFT and OS/MVT Data Management for System Programmers
1	Maintaining the System Catalog
1	Introduction
2	Reading a Block From the Catalog
2	Reading a Block by Data Set Name (LOCATE and CAMLST NAME)
4	Reading a Block by Generation Data Set Name (LOCATE and CAMLST NAME)
6	Reading a Block by Alias (LOCATE and CAMLST NAME)
7	Reading a Block by Relative Block Address (LOCATE and CAMLST BLOCK)
9	Building and Deleting Indexes
9	Building an Index (INDEX and CAMLST BLDX)
10	Building a Generation Index (INDEX and CAMLST BLDG)
11	Deleting an Index (INDEX and CAMLST DLTX)
12	Assigning an Alias for an Index (INDEX and CAMLST BLDA)
13	Deleting an Alias for an Index (INDEX and CAMLST DLTA)
14	Connecting and Disconnecting Control Volumes
14	Connecting Control Volumes (INDEX and CAMLST LNKX)
15	Disconnecting Control Volumes (INDEX and CAMLST DRPX)
16	Working with Data Set Catalogs
17	Cataloging a Data Set When Index Levels Exit (CATALOG and CAMLST CAT)
18	Cataloging a Data Set by Creating Required Index Levels (CATALOG and CAMLST CATBX)
20	Uncataloging a Data Set While Retaining Index Levels (CATALOG and CAMLST UNCAT)
20	Uncataloging a Data Set and Removing Index Levels (CATALOG and CAMLST UCATDX)
21	Recataloging a Data Set (CATALOG and CAMLST RECAT)
23	Catalog Block Entries
23	Volume Index Control Entry
24	Index Control Entry
25	Index Link Entry and Index Pointer Entry
26	Data Set Pointer Entry
27	Volume Control Block Pointer Entry
28	Volume Control Block
29	Control Volume (CVOL) Pointer Entry

<i>Page</i>	
30	Control Volume Pointer Entry (OLD)
30	Alias Entry
31	Generation Index Pointer Entry
33	Maintaining the Volume Table of Contents
33	Introduction
34	Reading a DSCB by Name (OBTAIN and CAMLST SEARCH)
35	Reading a DSCB by Actual Device Address (OBTAIN and CAMLST SEEK)
36	Deleting a Data Set (SCRATCH and CAMLST SCRATCH)
38	Renaming a Data Set (RENAME and CAMLST RENAME)
43	Executing Your Own Channel Programs (EXCP)
43	Executing Channel Programs in System and Problem Programs
44	System Use of EXCP
45	Use of EXCP in Problem Programs
45	EXCP Operations in a Nonpageable Region
45	EXCP Requirements
45	Channel Program
46	Control Blocks
46	Input/Output Block (IOB)
46	Event Control Block (ECB)
46	Data Control Block (DCB)
47	Data Extent Block (DEB)
47	Channel Program Execution
47	Initiation of the Channel Program
48	Modification of a Channel Program During Execution
49	Completion of Execution
49	Interruption Handling and Error Recovery Procedures
50	Appendages
53	Page Fix (PGFX) and Start I/O (SIO) Appendage
53	Page Fix (PGFX) Appendage
53	Normal Page Fix List Processing
53	Extended (10-entry) Page Fix List Processing
54	Start I/O (SIO) Appendage
55	Program Controlled Interruption (PCI) Appendage
56	End-of-Extent Appendage
56	Channel End (CE) Appendage
57	Abnormal End (XCE) Appendage
58	Block Multiplexer Channel Programming Notes
60	Macro Specifications for Use With EXCP
60	DCB – Define Data Control Block for EXCP
61	Foundation Block Parameters
62	EXCP Interface Parameters
62	Foundation Block Extension and Common Interface Parameters
64	Device-Dependent Parameters
66	OPEN – Initialize Data Control Block
67	EXCP – Execute Channel Program

<i>Page</i>	
67	ATLAS – Assigning an Alternate Track and Copying Data from the Defective Track
68	Using ATLAS
69	Operation of the ATLAS Program
72	EOV – End of Volume
74	CLOSE – Restore Data Control Block
75	Control Block Fields
75	Input/Output Block Fields
77	Event Control Block Fields
77	Data Extent Block Fields
79	Using XDAP to Read and Write to Direct-Access Devices
79	Introduction
79	XDAP Requirements
80	Macro Specifications for Use With XDAP
80	DCB – Define Data Control Block
80	OPEN – Initialize Data Control Block
81	XDAP – Execute Direct-Access Program
82	EOV – End of Volume
83	CLOSE – Restore Data Control Block
83	Control Blocks Used with XDAP
83	Event Control Block
83	Input/Output Block
84	Direct-Access Channel Program
85	Conversion of Relative Block Address to Actual Device Address
87	Obtaining Sector Number of a Block on a Device With the RPS Feature
89	Password Protecting Your Data Sets
89	Introduction
90	PASSWORD Data Set Characteristics
91	Creating Protected Data Sets
91	Protection Feature Operating Characteristics
91	Termination of Processing
91	Volume Switching
92	Data Set Concatenation
92	SCRATCH and RENAME Functions
92	Counter Maintenance
92	Using the PROTECT Macro Instruction to Maintain the PASSWORD Data Set
93	PASSWORD Data Set Characteristics and Record Format When You Use the PROTECT Macro Instruction
93	Number of Records for Each Protected Data Set
93	Protection Mode Indicator
94	PROTECT Macro Specification
99	Return Codes From the PROTECT Macro
99	System Macro Instructions
99	Introduction

<i>Page</i>	
99	Mapping System Data Areas
99	IEFUCBOB – Mapping the UCB
100	IEFJFCBN – Mapping the JFCB
100	CVT – Mapping the CVT
100	Obtaining I/O Device Characteristics
100	DEVTYPE Macro Specification
101	Device Characteristics Information
102	Output for Each Device Type
104	Reading and Modifying a Job File Control Block
105	OPEN – Initialize Data Control Block for Processing the JFCB
105	RDJFCB – Read a Job File Control Block
107	Ensuring Data Security by Validating the Data Extent Block
108	DEBCHK – Macro Specification
110	Removing Queued Requests and Restoring the Requests
111	PURGE – Remove an RQE From a Queue
113	RESTORE – Return Purged IOBs to Queues
115	Adding a UCS Image or FCB Image to the System Image Library
115	Introduction
115	Adding a UCS Image to the Image Library
117	Adding an FCB Image to the Image Library
119	Index

FIGURES

<i>Page</i>	
23	Figure 1. The Volume Index Control Entry
24	Figure 2. The Index Control Entry
25	Figure 3. The Index Link and Index Pointer Entries
26	Figure 4. The Data Set Pointer Entry
27	Figure 5. The Volume Control Block Pointer Entry
28	Figure 6. The Volume Control Block
29	Figure 7. The Control Volume (CVOL) Pointer Entry
30	Figure 8. The Alias Entry
31	Figure 9. The Generation Index Pointer Entry
52	Figure 10. The Request Queue Element (RQE)
52	Figure 11. Entry Points, Returns, and Available Work Registers for the I/O Supervisor Appendages
61	Figure 12. Data Control Block for Format EXCP (After Open)
72	Figure 13. Error Locations and Return Codes if CCHH is in the Count Area Field
73	Figure 14. Error Locations and Return Codes if CCHHRKDD is in the Count Area Field
76	Figure 15. Input/Output Block Format
78	Figure 16. Event Control Block After Posting of Completion Code (EXCP)
84	Figure 17. Event Control Block After Posting of Completion Code (XDAP)
85	Figure 18. The XDAP Channel Programs
95	Figure 19. Parameter List for ADD Function
96	Figure 20. Parameter List for REPLACE Function
97	Figure 21. Parameter List for DELETE Function
97	Figure 22. Parameter List for LIST Function
98	Figure 23. Return Codes from the PROTECT Macro Instruction
110	Figure 24. Macro Definition, JCL, and Utility Statements for Adding the PURGE Macro to Your Macro Library
111	Figure 25. Macro Definition, JCL, and Utility Statements for Adding the RESTORE Macro to Your Macro Library
112	Figure 26. PURGE Parameter List
114	Figure 27. Purge Chain for Restoring IOBs

SUMMARY OF MAJOR CHANGES

New Device Support

OS/VS2 supports several devices that are not supported by OS/VS1. The devices are:

- IBM 3505 Card Reader
- IBM 3525 Card Punch
- IBM 3400 Magnetic Tape Units
- IBM 2305-1 Fixed-Head Storage Device

The information about these new devices does not apply to installations using OS/VS1.

DEB Validity Checking

For installations using OS/VS2, a data set security feature has been added to the system control program to prevent inadvertant or malicious access by one user to another user's data. The macro specifications for the DEBCHK macro instruction is presented in the "System Macro Instructions" chapter. Information about this feature is provided to installations using OS/VS1 for planning purposes only.

Authorized Program Facility (APF)

The APF security-integrity feature in OS/VS2 makes it impossible for an unauthorized program to update a volume table of contents on a direct-access storage volume. Information on this restriction is provided in the "System Macro Instructions" chapter under the RDJFCB macro instruction. The APF feature is not available to installations using OS/VS1.

Editorial Changes

The chapter titled "Maintaining the Catalog and the VTOC" has been divided into two chapters "Maintaining the Catalog" and "Maintaining the VTOC." Macro specifications have been added to all chapters.

HOW OS/VS DIFFERS FROM OS/MFT and OS/MVT DATA MANAGEMENT FOR SYSTEM PROGRAMMERS

Page Fixing Using EXCP

The OS/VS relocation feature makes it necessary to (1) execute your program in a nonpageable region or (2) fix certain data areas and control blocks in real storage when executing your own channel programs. Significant changes were made to the chapter on executing your own channel programs using the EXCP macro instruction.

Direct-Access Device Support

OS/VS does not support five direct access devices that are supported by OS MFT/MVT. The devices are:

- IBM 2301 Drum
- IBM 2303 Drum
- IBM 2302 Disk Storage
- IBM 2311 Disk Storage
- IBM 2321 Data Cell Drive

MAINTAINING THE SYSTEM CATALOG

This chapter contains detailed information on how to maintain and modify the system catalog.

More detailed information about the catalog routines is available in *OS/VS Catalog Management Logic*.

Before using the information in this chapter, you should be familiar with the information contained in the following publications:

- *OS/VS and DOS/VS Assembler Language*, GC33-4010, which contains information you will need in order to code programs in the assembler language.
- *OS/VS Data Management Services Guide*, GC26-3783, which contains a general description of the structure of catalog indexes and generation data groups.
- *OS/VS Utilities*, GC35-0005, which tells how to use utility programs to maintain the system catalog.
- *OS/VS JCL Services*, GC28-0617, which tells how to catalog and uncatalog data sets using job control language statements.

Introduction

The catalog management routines that maintain and modify the catalog are called by the assembler language macro instructions presented in this chapter. These macros are most commonly used by the system control program or the IEHPROGM utility, but you may use them in your own routines.

Catalog management is a component of the OS/VS system control program that is used for keeping track of data sets when a program provides only the name of a cataloged data set. The catalog, itself a system data set (DSNAME=SYSCTLG), contains data set names correlated with the volume identification (volume serial number) and device type.

The physical organization of the catalog is the same as that of a partitioned data set directory. It is formatted into 256-byte blocks, containing variable-length entries. Data area layouts and detailed descriptions of the fields of each entry are provided in Figures 1 through 9 at the end of this chapter.

The functions you can perform using the catalog management macro instructions are:

- Reading a block from the catalog.
- Building an index.
- Building a generation index.
- Deleting an index.
- Assigning an alias.
- Deleting an alias.
- Connecting control volumes.
- Disconnecting control volumes.
- Cataloging a data set.
- Removing data set references from the catalog.
- Recataloging a data set.

Specifications for coding the macro instructions are presented with each function to be performed. Accompanying the descriptions are coding examples and programming notes; exceptional-return condition codes follow the coding examples. In the functional descriptions, offsets into data areas are numbered from zero (the first byte is byte zero).

Reading a Block From the Catalog

To read an entry from the catalog, use the LOCATE AND CAMLST macro instructions. You may specify the entry you want to read into your work area by using either (1) the fully or partially qualified name of a data set, or (2) the relative block address (TTR) of the block containing the entry. If you specify a fully qualified data set name, a list of volumes on which the data set resides will be read into your work area. This volume list always begins with a 1-byte entry that is count of the number of the number of volumes in the list. If the data set resides on more than 20 volumes, the address of a volume control block will follow the volume list entries.

If you specify a partially qualified data set name, the first block in the catalog corresponding to the lowest-level index specified will be read into your work area.

If you specify a relative block address (TTR), the block at that relative address in the catalog will be read into your work area.

See Figures 1 through 9 for descriptions of the contents of the volume control block and the other catalog data areas.

Reading a Block by Data Set Name (LOCATE and CAMLST NAME)

When you specify a data set name and the named data set resides on five or fewer volumes, a volume list is built in your work area. A volume list consists of an entry for each volume on which part of the data set resides; it is preceded by a 2-byte field that contains a count of the number of volumes in the list. The count field is followed by a variable number of 12-byte entries. Each 12-byte entry consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number.

If, however, the named data set resides on more than five volumes, a volume control block is read into your work area. A volume control block has essentially the same contents as a volume list, except that it can contain as many as 20 entries and can be linked to another volume control block (see Figure 6). The count field of each volume control block contains the remaining number of volume entries. For example, if a data set resides on 61 volumes, the count field would be decreased by 20 (61, 41, 21, 1) as you read each successive volume control block into your area. The first two bytes of the block contain the number of volume pointers for the data set. Each volume pointer is a 12-byte field that contains a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number.

For VS1 systems, device codes are presented in *OS/VS1 System Data Areas* in the section "The UCBTYP Field in the UCB."

For VS2 systems, device codes are presented in *OS/VS2 System Data Areas* in the section "The UCBTYP Field in the UCB."

If the named data set is stored on more than 20 volumes, bytes 252-254 of the block contain the relative track address of the next volume control block for the named data set; the last block has binary zeros in bytes 252-254. Byte 255 contains a binary zero.

If the named data set is stored on only one volume, bytes 242-243 of your area contain the relative track address of the DSCB for that data set; otherwise these bytes are zero. Byte 255 contains a binary zero.

The format is:

<i>[symbol]</i>	LOCATE	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	NAME,dsname-relexp,[cvol-relexp],area-relexp

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

NAME

this operand must be coded as shown in order to read a block from the catalog by name.

dsname-relexp

specifies the virtual storage location of a fully qualified data set name. The area that contains the name must be 44 bytes long. The name may be defined by a C-type Define Constant (DC) instruction. -

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

area-relexp

specifies the virtual storage location of your 265-byte work area, which you must define. The work area must begin on a doubleword boundary. The first 256 bytes of the work area will contain a volume list or the volume control block that is read from the catalog, and the last 6 bytes will contain the serial number of the volume on which the block was found. If the data set resides on one volume, bytes 252-254 may contain the relative track address of the DSCB. This address is relative to the beginning of the VTOC.

Example: In the following example, the catalog entry containing a list of the volumes on which data set A.B resides is read into virtual storage. The search for the catalog entry starts on the system residence volume.

	LOCATE	INDAB	
	Check Exceptional Returns		
*			READ CATALOG ENTRY FOR
INDAB	CAMLST	NAME,AB,,LOCAREA	DATA SET A.B INTO VIRTUAL
AB	DC	CL44 'A.B'	STORAGE AREA NAMED LOCAREA.
LOCAREA	DS	0D	LOCAREA ALSO CONTAINS 3-BYTE
	DS	265C	TTR AND 6-BYTE SERIAL
*			NUMBER

The LOCATE macro instruction points to the CAMLST macro instruction. NAME, the first operand of CAMLST, specifies that the system is to search the catalog for an entry using the name of a data set. AB, the second operand, specifies the virtual storage location of a 44-byte area into which you have placed the fully qualified name of a data set. LOCAREA, the fourth operand, specifies a 265-byte area you have reserved in virtual storage.

After execution of these macro instructions, the 265-byte area contains: a volume list or a volume control block for the data set A.B and the 6-byte serial number of the volume on which the entry was found (in bytes 259-264). If data set A.B resides on only one volume, bytes 252-254 of your area may contain the relative track address of the DSCB for data set A.B (relative to the beginning of the volume).

If a code of 4 is returned in register 15 indicating that the required control volume (CVOL) was not mounted, bytes 259-264 of the work area will contain the volume serial number of this required volume. If LOCATE finds an old CVOL pointer entry, and the CVOL is not mounted, binary zeros will be returned in bytes 252-255 of the work area. However, if a new CVOL pointer entry is found, the 4-byte device code of the CVOL will be returned in those bytes.

Control will be returned to your program at the next executable instruction after the LOCATE macro instruction. If the block has been successfully read from the catalog, register 15 will contain zeros. Otherwise, register 15 will contain one of the following exceptional return codes.

Code	Interpretation
------	----------------

- | | |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 4 | Either the required control volume was not mounted, there is a closed chain of control volume pointers, or the specified volume does not contain a catalog data set (SYSCTLG). The work area contains the volume serial number (in bytes 259-264) and the device code of the volume, if available (in bytes 252-255). Your work area contains the last block that was searched. |
| 8 | One of the names of the qualified name was not found or an unidentified entry was found. Register 0 contains the number of the last valid name in the qualified name. For example, if the qualified name A.B.C.D were specified, but name C did not exist at the level specified, register 0 would contain the binary number 2. The work area contains the serial number of the volume containing the index (in bytes 259-264).* |
| 12 | Either an index, an alias, or a control volume pointer was found when the list of qualified names was exhausted.* If an index pointer entry was found, the work area contains the first block of the specified index. |
| 16 | A data set resides at some level of the index other than the lowest index level specified. The work area contains the serial number of the volume containing the index in which a data set was found (in bytes 259-264).* |
| 20 | A syntax error exists in the name (for example, nine characters, a double delimiter, blank name field, or a qualified name when a simple name is needed). |
| 24 | A permanent I/O error was found when processing the catalog. |
| 28 | Relative track address (TTR) supplied to LOCATE is out of the SYSCTLG data set extents.* |
| 32 | Invalid work area pointer (for example, not a doubleword boundary). |

* Register 0 contains the number of index levels that were searched before the failure was encountered.

Reading a Block by Generation Data Set Name (LOCATE and CAMLST NAME)

You specify the name of a generation data set by using the fully qualified generation index name and the relative generation number of the data set. The value of a relative

generation number reflects the position of a data set in a generation data group. The following values can be used:

- **Zero** – specifies the latest data set cataloged in a generation data group.
- **Negative number** – specifies a data set cataloged before the latest data set.
- **Positive number** – specifies a data set not yet cataloged in the generation data group.

When you use zero or a negative number as the relative generation number, a volume list or volume control block (depending on whether these are more than five volumes in the data set) is read into virtual storage and the relative generation number is replaced by the absolute generation name.

When you use a positive number as the relative generation number, an absolute generation name is created and replaces the relative generation number. Nothing is read into your work area, because there are no entries in the catalog data sets.

The format is:

<i>[symbol]</i>	LOCATE	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	NAME,dsname-relexp,[cvol-relexp],area-relexp

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

NAME

this operand must be coded as shown in order to read a block from the catalog by generation data set name.

dsname-relexp

specifies the virtual storage location of the name of the generation index and the relative generation number. The area that contains these must be 44 bytes long. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

area-relexp

specifies the virtual storage location of your 265-byte work area, which you must define. The work area must begin on a doubleword boundary. The first 256 bytes of the work area will contain a volume list or the volume control block that is read from the catalog, and the last 6 bytes will contain the serial number of the volume on which the block was found. If the data set resides on one volume, bytes 252-254 may contain the relative track address of the DSCB. This address is relative to the beginning of the volume.

Example: In the following example, the list of volumes that contain generation data set A.PAY(-3) is read into virtual storage. The search for the catalog entry starts on the system residence volume.

The LOCATE macro instruction points to the CAMLST macro instruction. NAME, the first operand of CAMLST, specifies that the system is to search the catalog for a catalog entry by using the name of a data set. APAY, the second operand, specifies the virtual storage location of a 44-byte area into which you have placed the name of

	LOCATE	INDGX	READ CATALOG ENTRY FOR
	Check Exceptional Returns		
INDGX	CAMLST	NAME, APAY, , LOCAREA	DATA SET A.PAY(-3) INTO YOUR
APAY	DC	CL44 'A.PAY(-3)'	STORAGE AREA NAMED LOCAREA.
LOCAREA	DS	0D	LOCAREA ALSO CONTAINS 6-BYTE
	DS	265C	VOLUME SERIAL NUMBER

the generation index and the relative generation number of a data set in the generation data group. LOCAREA, the fourth operand, specifies a 265-byte area you have reserved to receive the catalog information.

After execution of these macro instructions, your 265-byte area contains: the catalog entry for generation data set A.PAY(-3) and the 6-byte serial number of the volume on which the block was found (in bytes 259-264). If data set A.PAY(-3) resides on one volume, bytes 252-254 of your area may contain the relative track address of the DSCB for that data set (relative to the beginning of the VTOC). In addition, the system will have replaced the relative generation number that you specified in your 44-byte area with the data set's absolute generation name. Control will be returned to your program at the next executable instruction after the LOCATE macro instruction. If the index block has been located and read successfully, register 15 will contain zeros. Otherwise, register 15 will contain one of the exceptional return codes described in the previous example.

Reading a Block by Alias (LOCATE and CAMLST NAME)

For each of the preceding functions, you can specify an alias as the first name in the qualified name of an index level, data set, or generation data set. Each function is performed exactly as previously described, with one exception: the alias name specified is replaced by the true name. Be aware, however, that if the true name of the data set is longer than the alias, the fully qualified name may exceed 44 characters.

The format is:

<i>symbol</i>]	LOCATE	<i>list-addrx</i>
[<i>list-name</i>]	CAMLST	NAME,dsname-relexp,[cvol-relexp],area-relexp

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

NAME

this operand must be coded as shown in order to read a block from the catalog by name.

dsname-relexp

specifies the virtual storage location of a fully qualified data set name, the first or only name of which is the alias. The area that contains the name must be 44 bytes long. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

area-relexp

specifies the virtual storage location of your 265-byte work area, which you must define. The work area must begin on a doubleword boundary. The first 256 bytes of the work area will contain a volume list or the volume control block that is read from the catalog, and the last 6 bytes will contain the serial number of the volume on which the block was found. If the data set resides on one volume, bytes 252-254 may contain the relative track address of the DSCB. This address is relative to the beginning of the VTOC.

Example: In the following example, the catalog entry containing a list of the volumes on which data set A.B.C resides is read into virtual storage. (Data set A.B.C, however, is addressed by an alias name -X is an alias for A.) The search for the catalog entry starts on the system residence volume.

LOCATE	INDAB	READ CATALOG	ENTRY FOR
Check Exceptional Returns			
INDAB	CAMLST	NAME,ABC,,LOCAREA	DATA SET X.B.C INTO VIRTUAL
ABC	DC	CL44 'X.B.C.'	STORAGE AREA NAMED LOCAREA.
LOCAREA	DS	0C	LOCAREA ALSO CONTAINS
	DS	265C	3-BYTE TTR AND 6-BYTE
*			SERIAL NUMBER

The LOCATE macro instruction points to the CAMLST macro instruction. NAME, the first operand of CAMLST, specifies that the system is to search the catalog for an entry using the name of a data set. ABC, the second operand, specifies the virtual storage location of a 44-byte area into which you have placed the fully qualified name of a data set. (In this case, data set A.B.C is addressed by its alias X.B.C.) LOCAREA, the fourth operand, specifies a 265-byte area you have reserved in virtual storage.

After execution of these macro instructions, the 265-byte area contains: a volume list or a volume control block for the data set A.B.C and the 6-byte serial number of the volume on which the entry was found (in bytes 259-264). If data set A.B.C resides on only one volume, bytes 252-254 of your area may contain the relative track address of the DSCB for data set A.B.C (relative to the beginning of the VTOC).

If a code of 4 is returned in register 15 indicating that the required control volume (CVOL) was not mounted, bytes 259-264 of the work area will contain the volume serial number of this required volume. If LOCATE finds an old CVOL pointer entry, and the CVOL is not mounted, binary zeros will be returned in bytes 252-255 of the work area. However, if a new CVOL pointer entry is found, the 4-byte device code of the CVOL will be returned in those bytes.

Reading a Block by Relative Block Address (LOCATE and CAMLST BLOCK)

You can read any block in the catalog by specifying, in the form TTR, the identification of the block and its location relative to the beginning of the catalog. TT is the number of tracks from the beginning of the catalog, R is the record number of the desired block on the track.

The format is:

<i>[symbol]</i>	LOCATE	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	BLOCK , <i>ttr-relexp</i> , <i>[cvol-relexp]</i> , <i>area-relexp</i>

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

BLOCK

you must code this operand as shown.

ttr-relexp

specifies the virtual storage location of a 3-byte relative block address (TTR). This address indicates the position relative to the beginning of the catalog data set, of the track containing the block (TT), and the block identification (R) on that track.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

area-relexp

specifies the virtual storage location of your 265-byte work area, which you must define. The work area must begin on a doubleword boundary. The first 256 bytes of the work area will contain the block that is read from the catalog, and the last 6 bytes will contain the serial number of the volume on which the block was found. If the data set resides on one volume, bytes 252-254 will contain the relative track address of the DSCB.

Example: In the following the block at the location indicated by TTR is read into virtual storage. The specified block is in the catalog on the system residence volume.

	LOCATE	BLK	
			Check Exceptional Returns
BLK	CAMLST	BLOCK,TTR,,LOCAREA	READ A BLOCK INTO VIRTUAL STORAGE. AREA NAMED LOCAREA
*			
TTR	DC	H'5'	RELATIVE TRACK 5
	DC	X'03'	BLOCK 3 ON TRACK
LOCAREA	DS	0D	LOCAREA ALSO CONTAINS 6-BYTE SERIAL NO.
	DS	265C	

The LOCATE macro instruction points to the CAMLST macro instruction. BLOCK, the first operand of CAMLST, specifies that the system is to search the catalog for the block indicated by TTR, the second operand. LOCAREA, the fourth operand, specifies a 265-byte area you have reserved in virtual storage.

After execution of these macro instructions, the 265-byte area contains: the 256-byte block and the 6-byte serial number of the volume on which the block was found (in bytes 259-264).

Control will be returned to your program at the next executable instruction following the LOCATE macro instruction. If the index block at the address you specified as been successfully located and read into your work area, register 15 will contain zeros. Otherwise, register 15 will contain one of the exceptional return codes described with the first example in this section.

Building and Deleting Indexes

You handle indexes – build them, delete them, etc. – by using combinations of the INDEX and CAMLST macro instructions.

Building an Index (INDEX and CAMLST BLDX)

To build a new index structure and add it to the catalog, you may create each level of the index separately. (You can also create index levels while you are cataloging a data set onto those index levels. See “Cataloging When Index Levels Exist (CATALOG and CAMLST CAT)” and “Cataloging by Creating Required Index Levels (CATALOG and CAMLST BLDX).”) To create each level of the index, use the INDEX and CAMLST macro instructions.

These two macro instructions can also be used to add index levels to existing index structures.

The format is:

<i>symbol</i>]	INDEX	<i>list-addrx</i>
[<i>list-name</i>]	CAMLST	BLDX , <i>name-relexp</i> , [<i>cvol-relexp</i>]

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

BLDX

this operand must be coded as shown.

name-relexp

specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by blanks. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

Example: In the following example, index structure A.B.C is built on the control volume whose serial number is 000045.

INDEX	INDEXA	BUILD	INDEX	A
Check Exceptional Returns				
INDEX	INDEXB	BUILD	INDEX	STRUCTURE A.B
Check Exceptional Returns				
INDEX	INDEXC	BUILD	INDEX	STRUCTURE A.B.C
Check Exceptional Returns				
INDEXA	CAMLST	BLDX, ALEVEL, VOLNUM		
INDEXB	CAMLST	BLDX, BLEVEL, VOLNUM		
INDEXC	CAMLST	BLDX, CLEVEL, VOLNUM		
VOLNUM	DC	CL6 '000045'	VOLUME SERIAL NUMBER	
ALEVEL	DC	CL2 'A'	INDEX STRUCTURE NAMES	
BLEVEL	DC	CL4 'A.B'	FOLLOWED BY A BLANK	
CLEVEL	DC	CL6 'A.B.C'	WHICH DELIMITS FIELDS	

Each INDEX macro instruction points to an associated CAMLST macro instruction. BLDX, the first operand of CAMLST, specifies that an index level be built. The second operand specifies the virtual storage location of the area into which you have placed the fully qualified name of an index level. The third operand specifies the virtual storage location of the area into which you have placed the 6-byte serial number of the volume on which the index level is to be built.

Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the index has been built successfully, register 15 will contain zeros. Otherwise, register 15 will contain one of the following exceptional return codes.

Code Interpretation

- 4 Either the required control volume was not mounted, or the specified volume does not contain a catalog data set (SYSCTLG).
- 8 The existing catalog structure is inconsistent with the operation performed. (Because the INDEX macro instruction uses the search routine of the LOCATE macro instruction, register 1 contains the condition code that would be given by the LOCATE macro instruction, and register 0 contains the number of index levels referred to during the search.)
- 12 An attempt was made to delete an index or generation index that has an alias or has indexes or data sets cataloged under it. The index is unchanged.
- 16 The qualified name specified when building an index or generation index implies an index structure that does not exist; the high level index, specified when connecting control volumes, does not exist.
- 20 Space is not available on the specified control volume.
- 24 Not used with the INDEX macro instruction.
- 28 A permanent I/O error was found when processing the catalog.
- 72 The VTOC of a DOS volume could not be converted to OS format.

Building a Generation Index (INDEX and CAMLST BLDG)

You build a generation index by using the INDEX and CAMLST macro instructions. All higher levels of the index must exist. If the higher levels of the index are not in the catalog, you must build them. How to build an index has been explained previously.

The format is:

<i>[symbol]</i>	INDEX	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	BLDG , <i>name-relexp</i> , <i>[cvol-relexp]</i> ,, <i>[DELETE]</i> , <i>[EMPTY]</i> , <i>number-absexp</i>

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

BLDG

this operand must be coded as shown.

name-relexp

specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by blanks. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

DELETE

specifies that all data sets on direct-access volumes that are removed from a generation data group are to be deleted, that is, the space allocated to the data set(s) is to be made available for reallocation. A SCRATCH macro instruction will be issued by the catalog management routines to delete the data set, which will be deleted from the volume if there are no conditions preventing deletion (e.g., expiration date not passed, password not verified, volume not mounted, permanent I/O error encountered while trying to delete the data set).

EMPTY

specifies that references to all data sets in a generation data group cataloged in the generation index are to be removed from the index when the number of entries specified is exceeded.

Number - absexp

specifies the number of data sets to be included in a generation data group. This number must be specified, and cannot exceed 255.

Example: In this example, generation index D is built on the control volume, serial number 000045. The higher level indexes A.B.C already exist. When the number of generation data sets in the generation index D exceeds four, the oldest data set is uncataloged. When the data set has been successfully uncataloged and the DELETE operand has been specified, the catalog management routines issue a SCRATCH macro (see "Maintaining the Volume Table of Contents") to delete the data set. If there are no conditions preventing the data set from being deleted (for example, the expiration date was not passed, the password could not be verified, or a permanent I/O error was encountered when trying to delete the data set), the data set will be deleted.

	INDEX	GENINDX	BUILD	GENERATION	INDEX
	Check Exceptional Returns				
GENINDX	CAMLST	BLDG,DLEVEL,VOLNUM,,DELETE,,4			
DLEVEL	DC	CL8'A.B.C.D'	ONE	BLANK,	DELIMITER
VOLNUM	DC	CL6'000045'			

The INDEX macro instruction points to the CAMLST macro instruction. BLDG, operand of CAMLST, specifies that a generation index be built. DLEVEL specifies the virtual storage location of an area into which you have placed the fully qualified name of a generation index. VOLNUM specifies the virtual storage location of the area into which you have placed the 6-byte serial number of the volume on which the generation index is to be built. DELETE specifies that all data sets dropped from the generation data group are to be deleted. The final operand, 4, specifies the number of data sets that are to be maintained in the generation data group. Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the generation index was built successfully, register 15 contains zeros. Otherwise, register 15 will contain one of the exceptional return codes described under "Building an Index (INDEX and CAMLST BLDX)."

Deleting an Index (INDEX and CAMLST DLTX)

You can delete any number of index levels from an existing index structure. Each level of the index is deleted separately. Generation indexes are also removed this way. (You can also delete index levels automatically when they are no longer needed. See

“Uncataloging a Data Set While Retaining Index Levels (CATALOG and CAMLST UNCAT)” and “Uncataloging a Data Set and Removing Index Levels (CATALOG and CAMLST UCATDX)” in this chapter for details). You delete each level of the index by using the INDEX and CAMLST macro instructions.

If an index level either has an alias, or has other index levels or data sets cataloged under it, it cannot be deleted.

The format is:

<i>[symbol]</i>	INDEX	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	DLTX , <i>name-relexp</i> , <i>[cvol-relexp]</i>

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

DLTX

this operand must be coded as shown.

name-relexp

specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by blanks. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

Example: In the following example, index level C is deleted from index structure A.B.C. The search for the index level starts on the system residence volume.

*	INDEX	DELETE	DELETE INDEX LEVEL C FROM
			INDEX STRUCTURE A.B.C
	Check Exceptional Returns		
DELETE	CAMLST	DLTX, LEVELC	
LEVELC	DC	CL6 'A.B.C'	ONE BLANK FOR DELIMITER

The INDEX macro instruction points to the CAMLST macro instruction. DLTX, the first operand of CAMLST, specifies that an index level be deleted. LEVELC, the second operand, specifies the virtual storage location of the area into which you have placed the fully qualified name of the index structure whose lowest level is to be deleted. Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the index level(s) was successfully deleted, register 15 contains zeros. Otherwise, register 15 contains one of the exceptional return codes described in “Building an Index (INDEX and CAMLST BLDX).”

Assigning an Alias for an Index (INDEX and CAMLST BLDA)

You assign an alias to an index level by using the INDEX and CAMLST macro instructions. An alias can be assigned only to a high level index; e.g., index A of index structure A.B.C can have an alias, but index B cannot. Assigning an alias to a high level index effectively provides aliases for all data sets cataloged under that index. An alias cannot be assigned to a generation index with only one level.

The format is:

<i>[symbol]</i>	INDEX	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	BLDA , <i>index name-relexp</i> , <i>[cvol-relexp]</i> , <i>alias name-relexp</i>

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

BLDA

this operand must be coded as shown.

index name-relexp

specifies the virtual storage location of the name of a high-level index. The area that contains the name must be 8 bytes long. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

alias name-relexp

specifies the virtual storage location of the name that is to be used as an alias for a high-level index. The area that contains the name must be 8 bytes long. The name may be defined by a C-type Define Constant (DC) instruction.

Example: In the following example, index level A is assigned an alias of X. The search for the index level starts on the system residence volume.

	INDEX	ALIAS	BUILD AN ALIAS FOR A HIGH LEVEL INDEX
*			
	Check Exceptional Returns		
ALIAS	CAMLST	BLDA,DSNAME,,DSALIAS	
DSNAME	DC	CL8 'A'	MUST BE 8-BYTE FIELDS
DSALIAS	DC	CL8 'X'	

The INDEX macro instruction points to the CAMLST macro instruction. BLDA, the first operand of CAMLST, specifies that an alias be built. DSNAME, the second operand, specifies the virtual storage location of an 8-byte area into which you have placed the name of the high-level index to be assigned an alias. DSALIAS, the fourth operand, specifies the virtual storage location of an 8-byte area into which you have placed the alias to be assigned.

Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the alias has been successfully assigned, register 15 will contain zeros. Otherwise, register 15 will contain one of the exceptional return codes described in "Building an Index (INDEX and CAMLST BLDX)."

Deleting an Alias for an Index (INDEX and CAMLST DLTA)

You can delete an alias previously assigned to a high level index by using the INDEX and CAMLST macro instructions.

The format is:

<i>[symbol]</i>	INDEX	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	DLTA , <i>alias name-relexp</i> , <i>[cvol-relexp]</i>

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

DLTA

this operand must be coded as shown.

alias name-relexp

specifies the virtual storage location of the name that is to be used as an alias for a high-level index. The area that contains the name must be 8 bytes long. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

Example: In the following example, alias X, previously assigned as an alias for index level A, is deleted. The search for the alias starts on the system residence volume.

	INDEX	DELALIAS	DELETE AN ALIAS FOR A
*			HIGH LEVEL INDEX
	Check Exceptional Returns		
DELALIAS	CAMLST	DLTA,ALIAS	
ALIAS	DC	CL8'X'	MUST BE 8-BYTE FIELD

The INDEX macro instruction points to the CAMLST macro instruction. DLTA, the first operand of CAMLST, specifies that an alias be deleted. ALIAS, the second operand, specifies the virtual storage location of the 8-byte area into which you have placed the alias to be deleted.

Connecting and Disconnecting Control Volumes

You connect and disconnect control volumes by using combinations of the INDEX and CAMLST macro instructions.

Connecting Control Volumes (INDEX and CAMLST LNKX)

You connect two control volumes (CVOLs) by using the INDEX AND CAMLST macro instructions. If a control volume is to be connected to the system residence volume, you need supply only the serial number of the volume to be connected and the name of a high level-index associated with the volume to be connected.

If a control volume is to be connected to a control volume other than the system residence volume, you must supply the serial numbers of both volumes and the name of a high-level index associated with the volume to be connected.

The result of connecting control volumes is that the volume serial number of the control volume connected and the name of a high-level index are entered into the volume index of the volume to which it was connected. This entry is called a control volume pointer.

The format is:

[symbol]	INDEX	<i>list-addrx</i>
[list-name]	CAMLST	LNKX, <i>index name-relexp</i> , <i>[cvol-relexp]</i> , <i>new cvol-relexp</i>

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

LNKX

this operand must be coded as shown.

index name-relexp

specifies the virtual storage location of the name of a high-level index. The area that contains the name must be 8 bytes long. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

new cvol-relexp

specifies the virtual storage location of the 4-byte device code and 6-byte volume serial number of the control volume that is to be connected to another control volume.

Example: In the following example, the control volume whose serial number is 001555 is connected to the control volume numbered 000155. The name of the high-level index is HIGHINDX.

	INDEX	CONNECT	CONNECT TWO CONTROL VOLUMES
	Check Exceptional Returns		
CONNECT	CAMLST	LNKX,INDXNAME,OLDCVOL	
*			WHOSE SERIAL NUMBERS ARE
INDXNAME	DC	CL8'HIGHINDX'	000155 AND 001555.
OLDCVOL	DC	CL6'000155'	
NEWCVOL	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'001555'	

The INDEX macro instruction points to the CAMLST macro instruction. LNKX, the first operand of CAMLST, specifies that control volumes be connected. INDXNAME, the second operand, specifies the virtual storage location of the 8-byte area into which you have placed the name of the high-level index of the volume to be connected. OLDCVOL, the third operand, specifies the virtual storage location of a 6-byte area into which you have placed the serial number of the volume to which you are connecting. NEWCVOL, the fourth operand, specifies the virtual storage location of a 10-byte area into which you have placed the 4-byte binary device code of the volume to be connected followed by the 6-byte area to contain the volume serial number of the volume to be connected.

Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the control volumes have been successfully connected, register 15 will contain zeros. Otherwise, register 15 will contain one of the exceptional return codes described in the section "Building an Index (INDEX and CAMLST BLDX)."

Disconnecting Control Volumes (INDEX and CAMLST DRPX)

You disconnect two control volumes by using the INDEX and CAMLST macro instructions. If a control volume is to be disconnected from the system residence volume, you need supply only the name of the high-level index associated with the volume to be disconnected.

If a control volume is to be disconnected from a control volume other than the system residence volume, you must supply, in addition to the name of the high-level index, the serial number of the control volume from which you want to disconnect.

The result of disconnecting control volumes is that the control volume pointer is removed from the volume index of the volume from which you are disconnecting.

The format is:

<i>[symbol]</i>	INDEX	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	DRPX,index name-relexp,[cvol-addrx]

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

DRPX

this operand must be coded as shown.

index name-relexp

specifies the virtual storage location of the name of a high-level index. The area that contains the name must be 8 bytes long. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

Example: In the following example, the control volume that contains the high-level index HIGHINDX is disconnected from the system residence volume.

*	INDEX	DISCONNECT	DISCONNECT TWO CONTROL VOLUMES
	Check Exceptional Returns		
DISCONNECT	CAMLST	DRPX,INDXNAME	
INDXNAME	DC	CL8 'HIGHINDX'	MUST BE 8-BYTE FIELD

The INDEX macro instruction points to the CAMLST macro instruction. DRPX, the first operand of CAMLST, specifies that control volumes be disconnected. INDXNAME, the second operand, specifies the virtual storage location of the 8-byte area into which you have placed the name of the high-level index of the control volume to be disconnected.

Control will be returned to your program at the next executable instruction following the INDEX macro instruction. If the control volumes were successfully disconnected, register 15 will contain zeros. Otherwise, register 15 will contain one of the exceptional return codes described in the section "Building an Index (INDEX and CAMLST BLDX)."

Working with Data Set Catalogs

You catalog, uncatalog, and recatalog data sets by using combinations of the CATALOG and CAMLST macro instructions.

When you catalog a data set, the CATALOG macro instruction points to the CAMLST macro instruction; parameters of the CAMLST macro instruction specify the options for cataloging a data set. When the CAT parameter is used, all index levels required to

catalog the data set must exist in the catalog. The index structure need not exist when the CATBX parameter is used; any missing index levels are automatically created. CATBX does not apply to generation indexes.

You must build a complete volume list in virtual storage. This volume list consists of an entry for each volume on which the data set is stored. The first two bytes of the list indicate the number of entries in the volume list; the number cannot be zero. Each 12-byte volume list entry consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. The sequence number is always zero for direct access volumes.

For VS1 systems, device codes are presented in *OS/VS1 System Data Areas* in the section "The UCBTYP Field in the UCB."

For VS2 systems, device codes are presented in *OS/VS2 System Data Areas* in the section "The UCBTYP Field in the UCB."

When you uncatalog or recatalog a data set, you use CATALOG and CAMLST in much the same way they are used in cataloging.

Cataloging a Data Set When Index Levels Exist (CATALOG and CAMLST CAT)

When the index levels already exist for a data set, you can use the CAT parameter of the CAMLST macro instruction to catalog the data set. Missing index levels cause an exceptional return code to be set.

The format is:

<i>[symbol]</i>	CATALOG	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	CAT , <i>name-relexp</i> , <i>[cvol-relexp]</i> , <i>vol list-relexp</i> , <i>[DSCBTTR=dscb ttr-relexp]</i>

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

CAT

this operand must be coded as shown.

name-relexp

specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by blanks. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

vol list-relexp

specifies the virtual storage location of an area that contains a volume list. The area must begin on a half-word boundary.

DSCBTTR= *dscb ttr-relexp*

specifies the virtual storage location of the 3-byte relative block address (TTR) of the format-1 data set control block (DSCB) for a data set that resides on only one volume. The address is relative to the beginning of the volume.

Example: In the following example, the data set named A.B.C is cataloged under an existing index structure A.B. The data set is stored on two volumes.

* *	CATALOG	ADDABC	CATALOG DATA SET A.B.C. THE INDEX STRUCTURE A.B EXISTS
	Check Exceptional Returns		
ADDABC	CAMLST	CAT,DSNAME,,VOLUMES	
DSNAME	DC	CL6'A.B.C'	ONE BLANK FOR DELIMITER
VOLUMES	DC	H'2'	DATA SET ON TWO VOLUMES
	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'000014'	VOLUME SERIAL NUMBER
	DC	H'0'	DATA SET SEQUENCE NUMBER
	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'000015'	VOLUME SERIAL NUMBER
	DC	H'0'	SEQUENCE NUMBER

The CATALOG macro instruction points to the CAMLST macro instruction. CAT, the first operand of CAMLST, specifies that a data set be cataloged. DSNAME, the second operand, specifies the virtual storage location of the area into which you have placed the fully qualified name of the data set to be cataloged. VOLUMES, the fourth operand, specifies the virtual storage location of the volume list you have built.

Control will be returned to your program at the next executable instruction following the CATALOG macro instruction. If your data set has been successfully cataloged, register 15 will contain zeros. Otherwise, register 15 will contain one of the following exceptional return codes.

Code Interpretation

- 4 Either the required control volume was not mounted, or the specified volume does not contain a catalog data set (SYSCTLG).
- 8 The existing catalog structure is inconsistent with the operation performed. (Because the CATALOG macro instruction uses the SEARCH option of the LOCATE macro instruction, register 1 contains the return code that would be returned by the LOCATE macro instruction. See the exceptional return considered as a result of the execution of a LOCATE macro under "Reading a Block from the Catalog." Register 0 contains the number of the index levels referred to before the exception was noted.)
- 12 Not used with the CATALOG macro instruction.
- 16 The index structure necessary to catalog the data set does not exist.
- 20 Space is not available on the specified control volume.
- 24 An attempt was made to catalog an improperly named generation data set, or the generation index is full and the named data set is older than any currently in the index.
- 28 A permanent I/O error was encountered when processing the catalog.
- 72 The VTOC of a DOS volume could not be converted to OS format.

Cataloging a Data Set by Creating Required Index Levels (CATALOG and CAMLST CATBX)

When index levels are missing, you can use the CATBX parameter of the CAMLST macro instruction to automatically create them before cataloging the data set.

The format is:

<i>[symbol]</i>	CATALOG	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	CATBX , <i>name-relexp</i> , <i>[cvol-relexp]</i> , <i>vol list-relexp</i> , <i>[DSCBTTR=dscb ttr-relexp]</i>

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

CATBX

this operand must be coded as shown.

name-relexp

specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by blanks. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

vol list-relexp

specifies the virtual storage location of an area that contains a volume list. The area must begin on a half-word boundary.

DSCBTTR= *dscb ttr-relexp*

specifies the virtual storage location of the 3-byte relative block address (TTR) of the identifier (format-1) DSCB for a data set that resides on only one volume. The block address is relative to the beginning of the volume.

Example: In the following example, the index structure A.B is created and data set A.B.C is cataloged. The data set is stored on one volume.

	CATALOG	CTBXABC	CATALOG DATA SET A.B.C.
*			CREATE NEEDED INDEX LEVELS
	Check Exceptional Returns		
CTBXABC	CAMLST	CATBX,DSNAME,,VOLUMES,DSCBTTR=TTR	
DSNAME	DC	CL6'A.B.C'	ONE BLANK FOR DELIMITER
VOLUMES	DC	H'1'	ONE VOLUME
	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'000015'	VOLUME SERIAL NUMBER
	DC	H'0'	DATA SET SEQUENCE NUMBER
TTR	DC	XL3'000103'	TTR OF DSCB RELATIVE TO
*			BEGINNING OF VTOC

The CATALOG macro instruction points to the CAMLST macro instruction. CATBX, the first operand of CAMLST, specifies that a data set is to be cataloged and any required higher level indexes are to be created. DSNAME, the second operand, specifies the virtual storage location of an area into which you have placed the fully qualified name of the data set to be cataloged. VOLUMES, the fourth operand, specifies the virtual storage location of the volume list you have built.

DSCBTTR=TTR, the fifth operand, specifies the virtual storage location into which you have placed the relative track address of the DSCB for the data set to be cataloged. The DSCBTTR operand is optional and is ignored for data sets residing on more than one volume.

Control will be returned to your program at the next executable instruction following the CATALOG macro instruction. If the index levels have been successfully created, register 15 will contain zeros. Otherwise, register 15 will contain one of the exceptional return codes described in the previous example.

Uncataloging a Data Set While Retaining Index Levels (CATALOG and CAMLST UNCAT)

When the UNCAT operand of the CAMLST macro instruction is used, a data set reference is removed, but all index levels are retained.

The format is:

<i>[symbol]</i>	CATALOG	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	UNCAT , <i>name-relexp</i> , <i>[cvol-relexp]</i>

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

UNCAT

this operand must be coded as shown.

name-relexp

specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by blanks. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

In the following example, references to data set A.B.C are removed from the catalog.

*	CATALOG	REMOVE	REMOVE REFERENCES TO DATA SET A.B.C FROM CATALOG
		Check Exceptional Returns	
REMOVE	CAMLST	UNCAT,DSNAME	
DSNAME	DC	CL6 'A.B.C'	ONE BLANK FOR DELIMITER

The CATALOG macro instruction points to the CAMLST macro instruction. UNCAT specifies that references to a data set be removed from the catalog. DSNAME specifies the virtual storage location of the area into which you have placed the fully qualified name of the data set whose references are to be removed.

Uncataloging a Data Set and Removing Index Levels (CATALOG and CAMLST UCATDX)

When the UCATDX operand of the CAMLST macro instruction is used, a data set reference and unneeded indexes, with the exception of the highest-level index, are removed from the catalog.

The format is:

[symbol]	CATALOG	list-addrx
[list-name]	CAMLST	UCATDX,name-relexp,[cvol-relexp]

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

UCATDX

this operand must be coded as shown.

name-relexp

specifies the virtual storage location of the fully qualified name of a data set or index level. The name cannot exceed 44 characters. If the name is less than 44 characters, it must be followed by blanks. The name may be defined by a C-type Define Constant (DC) instruction.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

Example: In the following example, references to data set A.B.C are removed from the catalog. Index B is removed unless it contains references to other data sets. Index A remains because it is the highest-level index.

*	CATALOG	RMDSNNDX		REMOVE REFERENCES TO DATA
		Check Exceptional Returns		SET A.B.C FROM CATALOG
	RMDSNNDX	CAMLST	UCATDX,DSNAME	AND REMOVE UNEEDED INDEXES
	DSNAME	DC	CL6 'A.B.C'	ONE BLANK FOR DELIMITER

The CATALOG macro instruction points to the CAMLST macro instruction. UCATDX, the first operand, specifies that references to a data set be removed from the catalog. DSNAME, the second operand, specifies the virtual storage location of the area into which you have placed the fully qualified name of the data set whose references are to be removed.

Control will be returned to your program at the next executable instruction following the CATALOG macro instruction. If the data set has been successfully uncataloged and its related index levels removed, register 15 will contain zeros. Otherwise, register 15 will contain one of the exceptional return codes described in the section titled "Cataloging a Data Set When Index Levels Exist (CATALOG and CAMLST CAT)."

Recataloging a Data Set (CATALOG and CAMLST RECAT)

You recatalog a cataloged data set by using the CATALOG and CAMLST macro instructions. Usually, a data set is recataloged when a new volume is added to the data set.

As in the original cataloging procedure, you must build a complete volume list in virtual storage. This volume list consists of an entry for each volume on which the data set resides. The first 2 bytes of the list indicate the number of entries in the list; the number may not be zero. Each 12-byte volume pointer consists of a 4-byte device

code, a 6-byte volume serial number, and a 2-byte data set sequence number. The sequence number is always zero for direct access volumes.

For VS1 systems, device codes are presented in *OS/VS1 System Data Areas* in the section "The UCBTYP Field in the UCB."

For VS2 systems, device codes are presented in *OS/VS2 System Data Areas* in the section "The UCBTYP Field in the UCB."

The format is:

[symbol]	CATALOG	<i>list-addrx</i>
[list-name]	CAMLST	RECAT , <i>name-relexp</i> , [<i>cvol-relexp</i>], <i>vol list-relexp</i> , [DSCBTTR = <i>dscb ttr-relexp</i>]

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

RECAT

this operand must be coded as shown.

cvol-relexp

specifies the virtual storage location of a 6-byte volume serial number for the volume to be processed. If this parameter is not specified, the system residence volume is processed.

vol list-relexp

specifies the virtual storage location of an area that contains a volume list. The area must begin on a half-word boundary.

DSCBTTR= *dscb ttr-relexp*

specifies the virtual storage location of the 3-byte relative track address (TTR) of the identifier (format-1) DSCB for a data set that resides on only one volume. The address is relative to the beginning of the volume.

Example: In the following example, the two-volume data set named A.B.C is recataloged to add a third volume. An entry is added to the volume list, which previously contained only two entries.

	CATALOG	RECATLG	RECATALOG DATA SET A.B.C
*			ADDING A NEW VOLUME
	Check Exceptional Returns		
RECATLG	CAMLST	RECAT,DSNAME,,VOLUMES	
DSNAME	DC	CL6'A.B.C'	POINTER TO THE VOLUME LIST.
*			FOR DELIMITER ONE BLANK
VOLUMES	DC	H'3'	THREE VOLUMES.
	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'000014'	VOLUME SERIAL NUMBER
	DC	H'0'	SEQUENCE NUMBER
	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'000015'	VOLUME SERIAL NUMBER
	DC	H'0'	SEQUENCE NUMBER
	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'000016'	VOLUME SERIAL NUMBER
	DC	H'0'	SEQUENCE NUMBER

The CATALOG macro instruction points to the CAMLST macro instruction. RECAT, the first operand of CAMLST, specifies that a data set be recataloged. DSNAME, the second operand, specifies the virtual storage location of an area into which you have

placed the fully qualified name of the data set to be recataloged. VOLUMES, the fourth operand, specifies the virtual storage location of the volume list you have built.

In this example, the entry for the new volume is added to the existing data set pointer entry by replacing the old volume list with the new volume list. If the total number of volumes in the data set had been increased to six or more, the data set pointer entry would have been replaced with a volume control block, which would contain an entry for each volume of the data set.

Control is returned to your program at the next executable instruction following the CATALOG macro instruction. If the data set has been successfully recataloged, register 15 will contain zeros. Otherwise, register 15 will contain one of the exceptional return codes described in the section "Cataloging a Data Set When Index Levels Exist (CATALOG and CAMLST CAT)."

Catalog Block Entries

This section describes the format and contents of each of the entries that may appear in the catalog.

Volume Index Control Entry

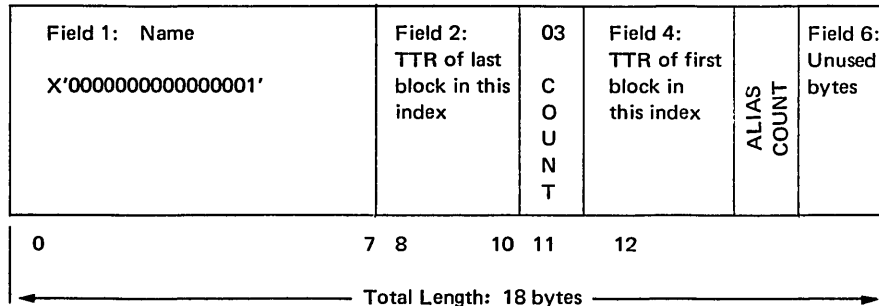
Field 1: Name X'0000000000000001'	Field 2: TTR of last block in volume index	05 C O U N T	Field 4: TTR of last block in SYSCTLG data set	00	Field 6: TTR of first unused block in SYSCTLG data set	00	Field 8: Unused bytes					
0	7	8	10	11	12	14	15	16	18	19	20	21
← Total Length: 22 bytes →												

The volume index control entry contains information about the entire catalog and the volume index. It is always the first entry in the catalog. It is 22 bytes long and contains eight fields.

- Field 1:** Name (8 bytes) – contains only a binary one to ensure that this entry is the first entry in the first block of the index.
- Field 2:** Last-block address (3 bytes) – contains the relative track address (TTR) of the last block in the volume index.
- Field 3:** Halfword count (1 byte) – contains a binary five to indicate that five halfwords follow.
- Field 4:** Catalog upper limit (3 bytes) – contains the relative track address (TTR) of the last block in the catalog data set.
- Field 5:** Zero field (1 byte) – contains binary zeros.
- Field 6:** First-available-block address (3 bytes) – contains the relative track address (TTR) of the unused block in the catalog that is closest to the beginning of the catalog data set.
- Field 7:** Zero field (1 byte) – contains binary zeros.
- Field 8:** Unused (2 bytes)

Figure 1. The Volume Index Control Entry

Index Control Entry



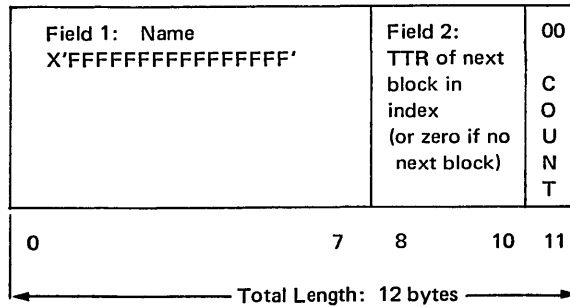
This index control entry is quite similar to a volume index control entry, but it only contains information about the index, which it begins. It is 18 bytes long and contains six fields.

- Field 1:** Name (8 bytes) – contains only a binary one to ensure that this entry, because it has the lowest binary name value, is the first entry in the first block of the index.
- Field 2:** Last block address (3 bytes) – contains the relative track address (TTR) of the last block assigned to this index.
- Field 3:** Halfword count (1 byte) – contains a binary three to indicate that 3 halfwords follow.
- Field 4:** Index lower limit (3 bytes) – contains the relative track address (TTR) of the block in which this entry appears.
- Field 5:** Number of aliases (1 byte) – contains the binary count of the number of aliases assigned to the index. If the index is not a high level index, this field is zero.
- Field 6:** Unused (2 bytes)

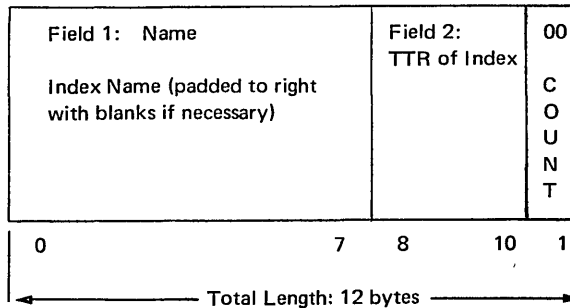
Figure 2. The Index Control Entry

Index Link Entry and Index Pointer Entry

Index Link Entry



Index Pointer Entry

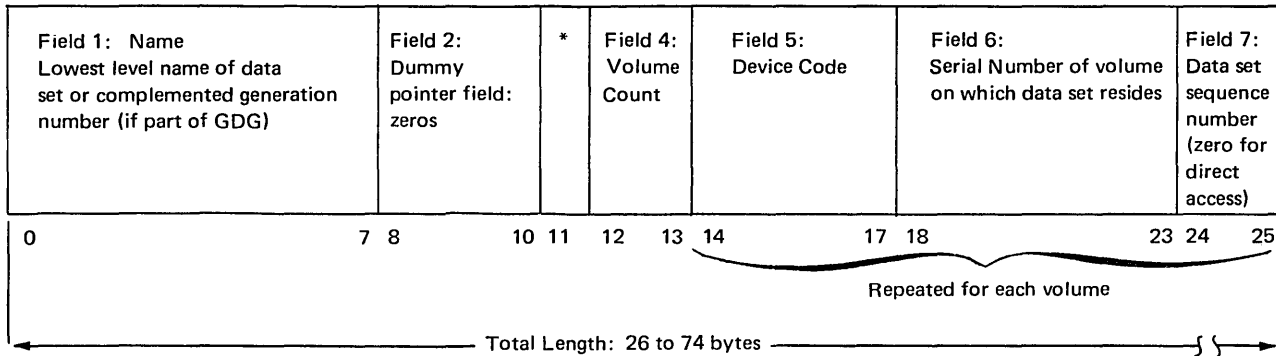


The index link and index pointer entries are quite similar. An index link entry is used to chain several blocks of an index together, and an index pointer entry is used to chain an index to the next lower level index. An index link entry is always the last entry in any index block. These blocks contain three fields and are 12 bytes long.

- Field 1:** Name (8 bytes) – contains the name of the index to which this entry points. If the entry is an index link entry, the name field contains X'FF FF FF FF FF FF FF FF'.
- Field 2:** Address (3 bytes) – contains either the relative block address (TTR) of the first block of the index if it is an index pointer entry, or the relative block address (TTR) of the next block of the index if it is an index link entry.
- Field 3:** Halfword count (1 byte) – contains 1 byte of binary zeros to indicate that the entry ends here.

Figure 3. The Index Link and Index Pointer Entries

Data Set Pointer Entry



* Count: equal to 6 times the number of volumes, plus 1.

The data set pointer entry can appear in any index. It contains the simple name of a data set and from one to five 12-byte fields, each of which identifies a volume on which the named data set resides. If the data set resides on more than five volumes, a volume control block pointer entry is substituted for the data set pointer entry. A volume control block pointer entry points to a volume control block or chain of volume control blocks that point to the volumes that contain the data set.

The data set pointer entry varies in length. The length is determined by the formula $14 + 12m$, where m is the number of volumes containing the data set. The variable m can be from one to five. The data set pointer entry can appear in any index, and it contains five fields.

Field 1: Name (8 bytes) – contains the simple name of the data set whose volumes are identified in field 5.

Field 2: Address (3 bytes) – this would normally be the address field, but since a data set pointer entry references no other entries in the catalog, it contains three bytes of binary zeros.

Field 3: Halfword count (1 byte) – contains the binary count of the number of halfwords that follow. The number is found by the formula $6m + 1$, where m is the number of volumes on which the data set resides. The variable m can be from one to five.

Field 4: Volume count (2 bytes) – contains the binary count of the number of volumes identified in field 5 of this entry.

Field 5: Device code (4 bytes) – contains the device code of the device on which the volume with the volume serial number in field 6 can be mounted.

For VS1 systems, device codes are presented in *OS/VS1 System Data Areas* in the section "The UCBTYP Field in the UCB."

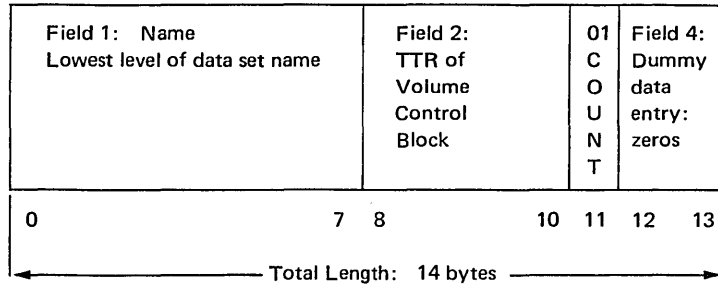
For VS2 systems, device codes are presented in *OS/VS2 System Data Areas* in the section "UCBTYP Field in the UCB."

Field 6: Volume serial number (6 bytes) – contains the volume serial number of one of the volumes of the data set.

Field 7: Volume sequence number (2 bytes) – contains the sequence number of the data set on a magnetic tape volume. It is zero for any other device class.

Figure 4. The Data Set Pointer Entry

Volume Control Block Pointer Entry

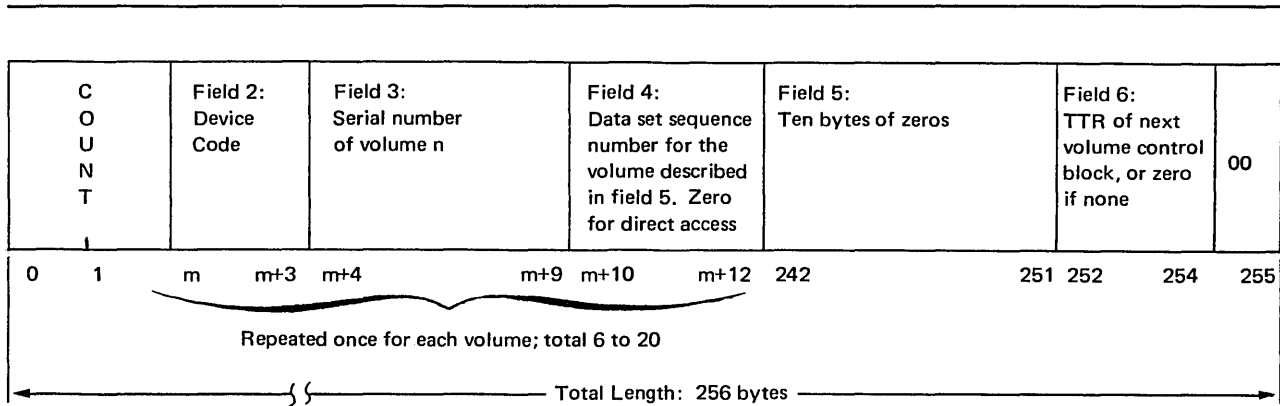


The volume control block pointer entry is used instead of a data set pointer entry when the data set resides on more than five volumes. This entry points to a volume control block, which, in turn, describes the data set. The entry is 14 bytes long.

- Field 1:** Name (8 bytes) – contains the last name of the qualified name of the data set identified by this entry.
- Field 2:** Address (3 bytes) – contains the relative block address (TTR) of the volume control block identifying the volumes containing the data set named in field 1.
- Field 3:** Halfword count (1 byte) – contains a binary one to indicate that one halfword follows.
- Field 4:** Zero field (2 bytes) – contains binary zeros.

Figure 5. The Volume Control Block Pointer Entry

Volume Control Block



A volume control block contains the description of all the volumes of a data set that resides on more than five volumes. One volume control block can describe as many as 20 volumes. Volume control blocks may be chained together to catalog a data set residing on more than 20 volumes.

The volume control block is always 256 bytes long, regardless of the number of volumes described.

Field 1: Volume count (2 bytes) – the first volume control block contains the binary count of the total number of volumes on which the data set resides. The value of this field is reduced by 20 for each subsequent volume control block. If, for example, the data set resides on 61 volumes, there will be four volume control blocks for the data set. The volume count field of each will contain 61, 41, 21, and 1, respectively.

Fields 2, 3, and 4: Volume identification (12 to 240 bytes) – contains from one to twenty 12-byte entries, each of which identifies a volume on which the data set resides. Each entry contains a 4-byte device code, a 6-byte volume serial number, and a 2-byte data set sequence number. The data set sequence number is zero for data sets on direct-access volumes.

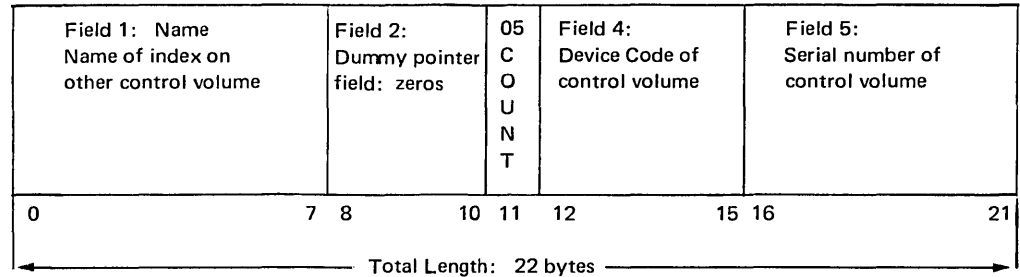
Field 5: Zero field (10 bytes) – contains binary zeros.

Field 6: Chain address (3 bytes) – contains the relative block address (TTR) of the next volume control block, if additional blocks are needed to describe the data set. If this is the last volume control block for the data set, this field will be set to binary zeros.

Field 7: Zero field (1 byte) – contains binary zeros.

Figure 6. The Volume Control Block

Control Volume (CVOL) Pointer Entry



NOTE: Prior to release 17, the control volume pointer entry contained a count of 03 and did not have a Device Code field (Field4)

Note: Prior to release 17, the control volume pointer entry contained a count of 03 and did not have a device code field (field 4).

The CVOL pointer entry is used to indicate that a particular index resides on a volume other than the system residence volume. Control volume pointer entries can exist only in the volume index. They are 22 bytes long.

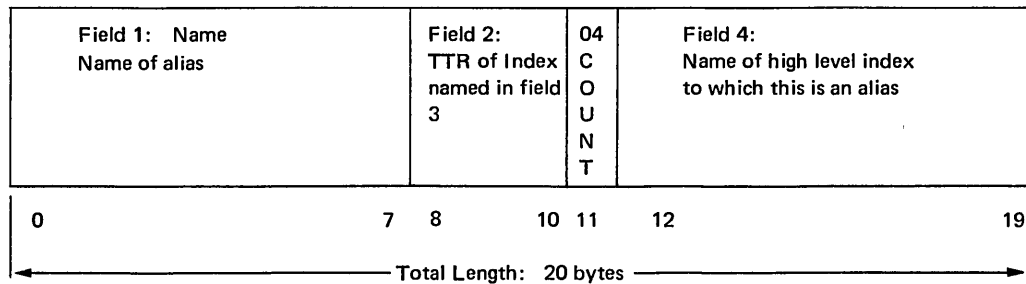
- Field 1:** Name (8 bytes) – contains a high-level index name that appears in the volume index of the control volume identified in fields 4 and 5.
- Field 2:** Address (3 bytes) – contains zeros, because this entry references no other entry in the catalog.
- Field 3:** Halfword count (1 byte) – contains the number 5 to indicate that five halfwords follow.
- Field 4:** CVOL device code (4 bytes) – This field contains the device code of the specified control volume.
- Field 5:** CVOL volume serial number (6 bytes) – contains the volume serial number of the control volume which has an entry in its volume index of the same name as this entry:

Figure 7. The Control Volume (CVOL) Pointer Entry

Control Volume Pointer Entry (Old)

Until Release 17 of OS MFT/MVT, the control volume pointer entry was the same as the present control volume pointer, except that there was no field 4 (device code). The old CVOL pointer entry was 18 bytes long; after Release 17, it is 22 bytes long. Since some control volumes may still contain entries in the old format, and since the catalog management routines still check for it, it is mentioned here.

Alias Entry

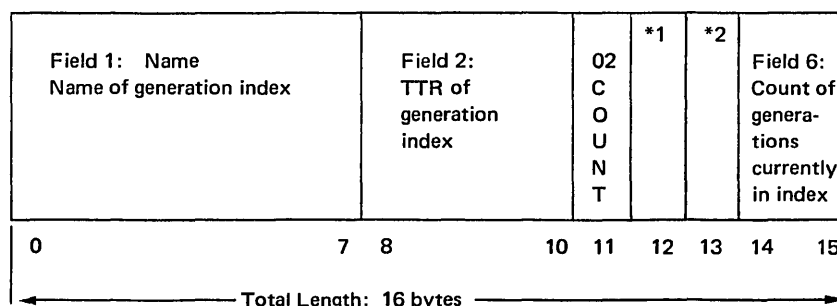


The alias entry is used to specify a substitute name for a high-level index. Alias entries only appear in the volume index. They are 20 bytes long.

- Field 1:** Name (4 bytes) – contains the alias of the high-level index identified in field 2.
- Field 2:** Address (3 bytes) – contains the relative block address (TTR) of the first block of the index named in field 4.
- Field 3:** Halfword count (1 byte) – contains a binary four to indicate that four halfwords follow.
- Field 4:** True name field (8 bytes) – contains the name of the index whose alias appears in field 1. The address of the index is in field 2.

Figure 8. The Alias Entry

Generation Index Pointer Entry



*1 Field 4:

Flags:	bits	meaning
	0-5	Reserved
	6	Delete
	7	Empty

*2 Field 5:
Count of maximum generations to be maintained in index

The generation index pointer entry points to a generation index. It is basically the same as an index pointer entry, except that it includes the flag and count fields. It is 16 bytes long.

Field 1: Name (8 bytes) – contains the lowest level name of the generation data group. That is, a generation data set named WEEKLY.INVNTRY.G0001V00 would have the name INVNTRY in the generation index pointer entry name field.

Field 2: Address field (3 bytes) – contains the relative block address (TTR) of the generation index named in field 1.

Field 3: Halfword count (1 byte) – contains a binary two to indicate that two halfwords follow.

Field 4: Flags (1 byte) – contains flags that govern the uncataloging of data sets as specified by the DELETE and EMPTY options of the INDEX macro instruction. The options and their hexadecimal codes are:

EMPTY = 01, DELETE = 02, and EMPTY and DELETE = 03; if no option was specified this byte is 00.

Field 5: Maximum number of generations allowed (1 byte) – contains the binary count of the maximum number of generations allowed in the index at one time, as specified in the INDEX macro instruction.

Field 6: Current generation count (2 bytes) – contains the binary count of the number of generations cataloged in the index.

Figure 9. The Generation Index Pointer Entry

MAINTAINING THE VOLUME TABLE OF CONTENTS

This chapter contains information on how to read and change the volume table of contents (VTOC) used on direct-access storage device volumes. The information consists of how-to information, macro specifications, and coding examples for the OBTAIN, SCRATCH, and RENAME macro instructions.

More detailed information about how the routines called by these macros work is available in *OS/VS DADSM Logic*, SY26-3787.

Before using the information in this chapter you should be familiar with the information contained in the following publications:

- *OS/VS and DOS/VS Assembler Language*, GC33-4010, which contains information you will need in order to code programs in the assembler language.
- *OS/VS Data Management Services Guide*, GC26-3783, contains a general description of direct-access device characteristics and the volume table of contents.
- *OS/VS Utilities*, GC35-0005, tells how to use utility programs to maintain the volume table of contents.
- *OS/VS1 System Data Areas*, SY28-0605, contains descriptions, (1) for OS/VS1, of the data set control block (DSCB) formats and (2) the contents of the fields of each DSCB.
- *OS/VS2 System Data Areas*, SY28-0606, which contains descriptions, for OS/VS2, of (1) the data set control block (DSCB) formats and (2) the contents of the fields of each DSCB.

Introduction

In the same way that the catalog management routines keep track of cataloged data sets, the direct-access device space management (DADSM) routines maintain the volume table of contents (VTOC) on direct-access storage devices. This chapter tells how to use the OBTAIN, SCRATCH, and RENAME macro instructions. These macros are most commonly used by the system control program and the data set utility programs (IEHMOVE, IEBCOPY, and IEHPROGM), but you may use them in your own routines. The functions you can perform with these macros are:

- Reading a data set control block from the VTOC – OBTAIN
- Deleting a data set – SCRATCH
- Changing the name of a data set – RENAME

You can read a data set control block (DSCB) into virtual storage by using the OBTAIN and CAMLST macro instructions. There are two ways to specify the DSCB that you want to read: by using the name of the data set associated with the DSCB, or by using the absolute track address of the DSCB. You must provide a 148-byte data area in virtual storage, into which the DSCB will be read. When you specify the name of the data set, an identifier (format-1) DSCB is read into virtual storage. To read a DSCB other than a format-1 DSCB, you must specify an absolute track address (see second example). (DSCB formats and field descriptions are contained in *OS/VS1 System Data Areas and OS/VS2 System Data Areas*.)

You can delete a data set by using the SCRATCH and CAMLST macro instructions. This causes the DSCBs for the data set to be deleted.

You can change a data set name by using the RENAME and CAMLST macro instructions. This causes the data set name in the identifier (format-1) DSCB for the data set to be replaced with new name.

Accompanying the descriptions of the macro instructions are coding examples, programming notes, and exceptional return code descriptions.

Reading a DSCB by Name (OBTAIN and CAMLST SEARCH)

When a data set name is specified, the 96-byte data portion of the identifier (format-1) DSCB, and the absolute track address of the DSCB are read into virtual storage. The absolute track address is a 5-byte field in the form CCHHR. When the absolute track address of a DSCB is specified, the 44-byte key portion and the 96-byte data portion of the DSCB are read into virtual storage, as shown in the second coding example.

[symbol]	OBTAIN	<i>list-addrx</i>
[list-name)	CAMLST	SEARCH , <i>dsname-relexp</i> , <i>vol-relexp</i> , <i>wkarea-relexp</i>

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

SEARCH

this operand must be coded as shown.

dsname-relexp

specifies the virtual storage location of a fully qualified data set name. The area that contains the name must be 44 bytes long. The name must be defined by a C-type Define Constant (DC) instruction.

vol-relexp

specifies the virtual storage location of the 6-byte volume serial number of the volume on which the DSCB is located.

wkarea-relexp

specifies the virtual storage location of a 148-byte work area that you must define. The work area must begin on a doubleword boundary.

Example: In the following example, the identifier (format-1) DSCB for data set A.B.C is read into virtual storage using the SEARCH option. The serial number of the volume containing the DSCB is 770655.

```

*          OBTAIN   DSCBABC          READ DSCB FOR DATA
*                                     SET A.B.C INTO DATA

          Check Exceptional Returns
DSCBABC   CAMLST   SEARCH,DSABC,VOLNUM,WORKAREA
DSABC     DC       CL44'A.B.C'       AREA NAMED WORKAREA.
VOLNUM    DC       CL6'770655'      96-BYTE DATA PORTION IS
WORKAREA  DS       0D                READ. THE REST OF THE AREA
*          DS       148C              IS USED BY THE OBTAIN
*                                     ROUTINE

```

The OBTAIN macro instruction points to the CAMLST macro instruction. SEARCH, the first operand of CAMLST, specifies that a DSCB be read into virtual storage, using the data set name you have supplied at the address indicated in the second operand. DSABC, the second operand, specifies the virtual storage location of a 44-byte area

into which you have placed the fully qualified name of the data set whose format-1 DSCB is to be read. VOLNUM, the third operand, specifies the virtual storage location of a 6-byte area into which you have placed the serial number of the volume containing the required DSCB. WORKAREA, the fourth operand, specifies the virtual storage location of a 148-byte work area into which the DSCB is to be read.

Control will be returned to your program at the next executable instruction following the OBTAIN macro instruction. If the DSCB has been successfully read into your work area, register 15 will contain zeros. Otherwise, register 15 will contain one of the following exceptional return codes:

Code Interpretation

- 4 The required volume was not mounted.
- 8 The format-1 DSCB was not found in VTOC of specified volume.
- 12 A permanent I/O error was found when processing the specified volume.
- 16 Invalid workarea pointer.

After execution of these macro instructions, the first 96 bytes of the work area contain the data portion of the identifier (format 1) DSCB; the next 5 bytes contain the absolute track address (CCHHR) of the DSCB.

Reading a DSCB by Actual Device Address (OBTAIN and CAMLST SEEK)

You can read any DSCB from a VTOC using the SEEK option.

The format is:

<i>[symbol]</i>	OBTAIN	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	SEEK, cchhr-relexp, vol-relexp, wkarea-relexp

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

SEEK

this operand must be coded as shown.

cchhr-relexp

specifies the virtual storage location of the 5-byte absolute device address (CCHHR) of a DSCB.

vol-relexp

specifies the virtual storage location of the 6-byte volume serial number of the volume on which the DSCB is located.

wkarea-relexp

specifies the virtual storage location of a 148-byte work area that you must define. The work area must begin on a doubleword boundary.

Example: In the following example, the DSCB at actual-device address X'00 00 00 01 07' is read into the virtual storage location READAREA, using the SEEK option. The DSCB resides on the volume with the volume serial number 108745.

The OBTAIN macro points to the CAMLST macro. SEEK, the first operand of CAMLST, specifies that a DSCB be read into virtual storage. MBCCCHHR, the second operand, specifies the storage location that contains the 5-byte actual-device

*	OBTAIN	ACTADDR	READ THE DSCB AT ACTUAL DEVICE INTO
Check Exceptional Returns			
ACTADDR	CAMLST	SEEK, CCHHR, VOLSER, READAREA	RETURNS
CCHHR	DC	XL5'0000000107'	STORAGE LOCATION NAMED
VOLSER	DC	CL6'108745'	WORKAREA. 140-BYTE DSCB IS
READAREA	DS	0D	READ. THE LAST 8 BYTES OF
*	DS	148C	READAREA IS USED BY THE OBTAIN ROUTINE.

address of the DSCB. VOLSER, the third operand specifies the storage location that contains the volume serial number of the volume on which the DSCB resides. The fourth operand, READAREA, specifies the storage location into which the 140-byte DSCB is to be read. The last 8 bytes are used by the OBTAIN routine.

Control will be returned to your program at the next executable instruction following the OBTAIN macro instruction. If the DSCB has been successfully read into your work area, register 15 will contain zeros. Otherwise, register 15 will contain one of the following exceptional return codes:

Code	Interpretation
4	The required volume was not mounted.
8	The format-1 DSCB was not found in VTOC of specified volume.
12	A permanent I/O error was found when processing the specified volume.
16	Invalid workarea pointer.

Deleting a Data Set (SCRATCH and CAMLST SCRATCH)

You delete a data set stored on direct-access volumes by using the SCRATCH and CAMLST macro instructions. This causes all data set control blocks (DSCBs) for the data set to be deleted, and all space occupied by the data set to be made available for reallocation. If the data set to be deleted is sharing one or more cylinders with one or more data sets (a split-cylinder data set), the space will not be made available for reallocation until all data sets on the shared cylinders are deleted.

A data set cannot be deleted if the expiration date in the identifier (format-1) DSCB has not passed, unless you choose to ignore the expiration date. You specify that the expiration date is to be ignored by using the OVRD option in the CAMLST macro instruction.

If a data set to be deleted is stored on more than one volume, either a device must be available on which to mount the volumes, or at least one volume must be mounted. In addition, all other required volumes must be serially mountable.

When deleting a data set, you must build a volume list in virtual storage. This volume list consists of an entry for each volume on which the data set resides. The first two bytes of the list indicate the number of entries in the list. Each 12-byte entry consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte scratch status code. Device codes are presented in *OS/VS1 System Data Areas* and in *OS/VS2 System Data Areas* in the sections titled "The UCBTYP Field of the UCB."

Volumes are processed in the order that they appear in the volume list. The volume at the beginning of the list is processed first. If a volume is not mounted, a message is

issued to the operator requesting him to mount the volume. This is only done if you indicate the direct access device on which unmounted volumes are to be mounted by loading register 0 with the address of the UCB associated with the device to be used. If you do not load register 0 with a UCB address, its contents must be zero, and at least one of the volumes in the volume list must be mounted before the SCRATCH macro instruction is issued.

If the operator cannot mount the requested volume, he issues a reply indicating that he cannot fulfill the request. A condition code is then set in the last byte of the volume pointer (the second byte of the scratch status code) for the unavailable volume, and the next volume indicated in the volume list is processed.

The format is:

<i>[symbol]</i>	SCRATCH	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	SCRATCH,dsname-relexp,vol list-relexp,,[OVRD]

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

SCRATCH

this operand must be coded as shown.

dsname-relexp

specifies the virtual storage location of a fully qualified data set name. The area that contains the name must be 44 bytes long. The name must be defined by a C-type Define Constant (DC) instruction.

vol list-relexp

specifies the virtual storage location of an area that contains a volume list. The area must begin on a half-word boundary.

OVRD

when coded as shown, specifies that the expiration date in the DSCB should be ignored.

Example: In the following example, data set A.B.C is deleted from two volumes. The expiration date in the identifier (format 1) DSCB is ignored.

	SR	0,0	SET REG 0 TO ZERO
	SCRATCH	DELABC	DELETE DATA SET
*			A.B.C. FROM TWO VOLUMES,
	Check Exceptional Returns		
DELABC	CAMLST	SCRATCH,DSABC,,VOLIST,,OVRD	
DSABC	DC	CL44'A.B.C'	IGNORING THE EXPIRATION
VOLIST	DC	H'2'	DATE IN THE DSCB.
	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'000017'	VOLUME SERIAL NO.
	DC	H'0'	SCRATCH STATUS CODE
	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'000018'	VOLUME SERIAL NO.
	DC	H'0'	SCRATCH STATUS CODE

The SCRATCH macro instruction points to the CAMLST macro instruction. SCRATCH, the first operand of CAMLST, specifies that a data set be deleted. DSABC, the second operand, specifies the virtual storage location of a 44-byte area

into which you have placed the fully qualified name of the data set to be deleted. VOLIST, the fourth operand, specifies the virtual storage location of the volume list you have built. OVRD, the sixth operand, specifies that the expiration date in the DSCB of the data set to be deleted be ignored.

When you attempt to delete a password-protected data set, the operating system issues a message (IEC301A) to ask the operator at the console or terminal operator of a remote console to enter the password. The data set will be scratched only if the password supplied is associated with a "WRITE" protection mode indicator. The protection word indicator is described in the chapter titled "Data Set Protection."

Control is returned to your program at the next executable instruction following the SCRATCH macro instruction. If the data set has been successfully deleted, register 15 will contain zeros and the scratch status code in the volume list entry for each volume will be set to zero. Otherwise, register 15 will contain one of the exceptional return codes that follow. To determine whether the data set has been successfully deleted from each volume on which it resides, you must examine the scratch status code, the last byte of each entry in the volume list.

**Code Interpretation
in
Reg. 15**

- 4 No volumes containing any part of the data set were mounted, nor did register 0 contain the address of a unit that was available for mounting a volume of data set.
- 8 An unusual condition was encountered on one or more volumes.

After the SCRATCH macro instruction is executed, the last byte of each 12-byte entry in the volume list indicates the following conditions in binary codes:

**Scratch
Status**

Code Interpretation

- 0 The DSCB for the data set has been deleted from the VTOC on the volume pointed to.
- 1 The VTOC of this volume does not contain the format-1 DSCB for the data set to be deleted.
- 2 The macro instruction failed when the correct password was not supplied in the two attempts allowed.
- 3 The data set was not deleted from this volume because either the OVRD option was not specified or the retention cycle has not expired.
- 4 A permanent I/O error was found when processing this volume.
- 5 A device for mounting this volume was unavailable.
- 6 The operator was unable to mount this volume.

Renaming a Data Set (RENAME and CAMLST RENAME)

You rename a data set stored on one or more direct-access volumes by using the RENAME and CAMLST macro instructions. This causes the data set name in all identifier (format-1) DSCBs for the data set to be replaced by the new name that you supply.

If a data set to be renamed is stored on more than one volume, either a device must be available on which to mount the volumes, or at least one volume must be mounted. In addition, all other volumes of the data set must be serially mountable.

When renaming a data set, you must build a volume list in virtual storage. This volume list consists of an entry for each volume on which the data set resides. The first two bytes of the list indicate the number of entries in the list. Each 12-byte volume list entry consists of a 4-byte device code, a 6-byte volume serial number, and a 2-byte rename status code. Device codes are presented in *OS/VS1 System Data Areas* and in *OS/VS2 System Data Areas* in the sections titled "The UCBTYP Field of the UCB." Volumes are processed in the order they appear in the volume list. The first volume on the list is processed first. If a volume is not mounted, a message is issued to the operator requesting him to mount the volume. This is only done if you indicate the direct-access device on which unmounted volumes are to be mounted by loading register 0 with the address of the UCB associated with the device to be used. If you do not load register 0 with a UCB address, its contents must be zero, and at least one of the volumes in the volume list must be mounted before the RENAME macro instruction is executed.

If the operator cannot mount a volume in the volume list, he issues a reply indicating that he cannot fulfill the request. A condition code is then set in the last byte of the volume list entry (the second byte of the rename status code) for the unavailable volume, and the next volume indicated in the volume list is processed or requested.

The format is:

<i>[symbol]</i>	RENAME	<i>list-addrx</i>
<i>[list-name]</i>	CAMLST	RENAME,dsname-relexp,new name-relexp,vol list-relexp

list-addrx

points to the parameter list (labeled *list-name*) set up by the CAMLST macro instruction.

RENAME

this operand must be coded as shown.

dsname-relexp

specifies the virtual storage location of a fully qualified data set name. The area that contains the name must be 44 bytes long. The name must be defined by a C-type Define Constant (DC) instruction.

new name-relexp

specifies the virtual storage location of a fully qualified data set name that is to be used as the new name. The area that contains the name must be 44 bytes long. The name must be defined by a C-type Define Constant (DC) instruction.

vol list-relexp

specifies the virtual storage location of an area that contains a volume list. The area must begin on a halfword boundary.

Example: In the following example, data set A.B.C is renamed D.E.F. The data set resides on two volumes.

The RENAME macro instruction points to the CAMLST macro instruction. RENAME, the first operand of CAMLST, specifies that a data set be renamed. OLDNAME, the second operand, specifies the virtual storage location of a 44-byte area into which you have placed the fully qualified name of the data set to be renamed.

	SR	0,0	SET REG 0 TO ZERO
*	RENAME	DSABC	CHANGE DATA SET NAME A.B.C. TO D.E.F

Check Exceptional Returns

DSABC	CAMLST	RENAME, OLDNAME, NEWNAME, VOLIST	
OLDNAME	DC	CL44 'A.B.C'	
NEWNAME	DC	CL44 'D.E.F'	
VOLIST	DC	H'2'	TWO VOLUMES
	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'000017'	VOLUME SERIAL NO.
	DC	H'0'	RENAME STATUS CODE
	DC	X'30C02008'	2314 DISK DEVICE CODE
	DC	CL6'000018'	VOLUME SERIAL NO.
	DC	H'0'	RENAME STATUS CODE

NEWNAME, the third operand, specifies the virtual storage location of a 44-byte area into which you have placed the new name of the data set. VOLIST, the fourth operand, specifies the virtual storage location of the volume list you have built.

Control is returned to your program at the next executable instruction following the RENAME macro instruction. If the data set has been successfully renamed, register 15 will contain zeros, and the rename status code in the volume list entry for each volume will be set to zero. Otherwise, register 15 will contain one of the exceptional return codes that follow. To determine whether the data set has been successfully renamed on each volume on which it resides, you must examine the rename status code, the last byte of each entry in the volume list.

Return Interpretation

Code in

Reg. 15

- 4 No volumes containing any part of the data set were mounted, nor did register 0 contain the address of a unit that was available for mounting a volume of the data set.
- 8 An unusual condition was encountered on one or more volumes.

After the RENAME macro instruction is executed, the last byte of each 12-byte entry in the volume list indicates the following conditions in binary code:

Rename

Status

Code Interpretation

- 0 The DSCB for the data set has been renamed in the VTOC on the volume pointed to.
- 1 The VTOC of this volume does not contain the format-1 DSCB for the data set to be renamed.
- 2 The macro instruction failed when the correct password was not supplied in the two attempts allowed.
- 3 A data set with the new name already exists on this volume.
- 4 A permanent I/O error was found when processing this volume.
- 5 A device for mounting this volumes was unavailable.
- 6 The operator was unable to mount this volume.

When you attempt to rename a password-protected data set, the operating system issues a message (IEC301A) to ask the operator or remote console operator to verify

the password. The data set will be renamed only if the password supplied is associated with a “WRITE” protection mode indicator. The chapter titled “Data Set Protection” provides a description of the protection mode indicator.

EXECUTING YOUR OWN CHANNEL PROGRAMS (EXCP)

The execute channel program (EXCP) macro instruction provides you with a device-dependent means of performing the I/O operations. This chapter contains a general description of the function and application of the EXCP macro instruction, accompanied by descriptions of specific control blocks and macro instructions used with EXCP. Factors that affect the operation of EXCP, such as device variations and program modification, are also discussed.

Before reading this chapter, you should be familiar with system functions and with the structure of control blocks, as well as with the operational characteristics of the I/O devices required by your channel programs. Operational characteristics of specific I/O devices are contained in IBM publications for each device.

To understand this chapter, you need to understand the information in these publications:

- *OS/VS Data Management Services Guide*, GC26-3783, explains the standard procedures for I/O processing under the operating system.
- *OS/VS and DOS/VS Assembler Language*, GC33-4010, contains the information necessary to code programs in the assembler language.
- *OS/VS Data Management Macro Instructions*, GC26-3793, describes the system macro instructions that can be used in programs coded in the assembler language.
- *OS/VS1 System Data Areas*, SY28-0605, contains format and field descriptions, for OS/VS1, of the system control blocks referred to in this chapter.
- *OS/VS2 System Data Areas*, SY28-0606, contains format and field descriptions, for OS/VS2, of the system control blocks referred to in this chapter.

The execute channel program (EXCP) macro instruction causes a supervisor-call interruption to pass control to the I/O supervisor. EXCP also provides the I/O supervisor with control information regarding a channel program to be executed. When the IBM standard data access methods are being used, the access method routines are responsible for issuing EXCP. If you are not using the standard access methods, you may issue EXCP in your program. Direct use of EXCP provides you with device dependence in organizing data and controlling I/O devices.

You issue EXCP primarily for I/O programming situations to which the standard access methods do not apply. When you are writing your own data access methods, you must include EXCP for I/O operations. EXCP must also be used for processing nonstandard labels, including reading and writing labels and positioning magnetic tape volumes.

To issue EXCP, you must provide a channel program (a list of channel command words) and several control blocks in your program area. The I/O supervisor then schedules I/O requests for the device you have specified, executes the specified I/O commands, handles I/O interruptions, directs error recovery procedures, and posts the results of the I/O requests.

Executing Channel Programs in System and Problem Programs

This section briefly explains the procedures performed by the system and the programmer when EXCP is issued by the routines of the standard data access methods.

The additional procedures that you must perform when issuing EXCP yourself are then described by direct comparison.

System Use of EXCP

When using a standard data access method to perform I/O operations, the programmer is relieved of coding channel programs and of constructing the control blocks necessary for the execution of channel programs. To permit I/O operations to be handled by an access method, the programmer need only issue the following macro instructions:

- A DCB macro instruction that produces a data control block (DCB) for the data set to be retrieved or stored. If appendages are not being used, a short DCB is constructed. Such a DCB does not support reduced error recovery.
- An OPEN macro instruction that initializes the data control block and produces a data extent block (DEB) for the data set.
- A macro instruction (e.g., GET, WRITE) that requests I/O operations.

Access method routines will then:

1. Create a channel program that contains channel commands for the I/O operations on the appropriate device.
2. Construct an input/output block (IOB) that contains information about the channel program.
3. Construct an event control block (ECB) that is later posted with a completion code each time the channel program terminates.
4. Issue an EXCP macro instruction to pass the address of the IOB to the routines that initiate and supervise the I/O operations.

The input/output supervisor will then:

5. Construct a request queue element (RQE) for scheduling the request.
6. If the requestor is in pageable partition, fix the pages to be referenced during the I/O operation so that they cannot be paged out; this includes pages for I/O control blocks and appendages.
7. If the requestor is in a pageable partition, translate the requestor's virtual channel program into a real channel program in the System Queue Area and fix the pages to be used as data areas during the I/O operation.
8. Issue a start input/output (SIO) instruction to cause the channel to execute the real channel program.
9. Process I/O interruptions and schedule error recovery procedures when necessary.
10. If the requestor is in a pageable partition, retranslate the "last CCW+8" address in the channel status word (CSW) to the corresponding virtual address.
11. If the requestor is in a pageable partition, free the real storage used for the channel program translation and unfix the pages that were fixed especially for the just completed I/O operation.
12. Post a completion code in the event control block after the channel program has been executed.

Note: If the requestor is in a nonpageable partition, he provides a real channel program, so items 6, 7, 10, and 11 are not performed.

The programmer is not concerned with these procedures and does not know the status of I/O operations until they are completed. Device-dependent operations are limited to those provided by the macro instructions of the particular access method selected.

Use of EXCP in Problem Programs

To issue the EXCP macro instruction directly, you must perform the procedures that the access methods perform, as summarized in items 1 through 4 of the preceding discussion. You must, in addition to constructing and opening the data control block with the DCB and OPEN macro instructions, construct a channel program, an input/output block, and an event control block before you can issue EXCP. The I/O supervisor always handles items 5 through 12.

After issuing EXCP, you should issue a WAIT macro instruction specifying the event control block to determine whether the channel program has terminated. If volume switching is necessary, you must issue an EOVS macro instruction. When processing of the data set has been completed, you must issue a CLOSE macro instruction to restore the data control block.

EXCP Operations in a Nonpageable Region

User-constructed channel programs for I/O operations in a nonpageable region are not translated. Because the partition is nonpageable, any CCWs created by the user have correct real data addresses. (Translation would only recreate the user's channel program, so the CCWs are used directly.)

System requests for I/O on behalf of a user, however, do not always operate in the user's nonpageable region. These requests make use of areas with a protection key of 0, which are pageable. Therefore, system requests for I/O on behalf of the user can require translation.

Modification of an active channel program by data read in or by CPU instructions is legitimate in a nonpageable region, but not in a pageable region. Refer to the section "Modification of a Channel Program During Execution".

EXCP Requirements

This section describes the channel program that you must provide in order to issue EXCP. The control blocks that you must either construct directly, or cause to be constructed by use of macro instructions, are also described.

Channel Program

The channel program supplied by you and executed through EXCP is composed of channel command words (CCWs) on doubleword boundaries. Each channel command word specifies a command to be executed and, for commands initiating data transfer, the area to or from which the data is to be transferred. Channel programs to be executed under VS1 in a pageable partition are restricted to no more than 240 CCWs. The I/O supervisor abnormally terminates an I/O requestor requiring translation of a channel program exceeding that limit. No such restriction applies to channel programs to be executed under VS2.

Channel command word formats used with specific I/O devices can be found in IBM Systems Reference Library publications for each device. All channel command words

described in these publications can be used, with the exception of REWIND and UNLOAD (RUN). In addition, both data chaining and command chaining may be used.

Chaining is the successive loading of channel command words into a channel from contiguous doubleword locations in real storage. Data chaining occurs when a new channel command word loaded into the channel defines a new storage area for the original I/O operation. Command chaining occurs when the new channel command word specifies a new I/O operation. For detailed information about chaining, refer to *IBM System/370 Principles of Operation, GA22-7000*.

To specify either data chaining or command chaining, you must set appropriate bits in the channel command word, and indicate the type of chaining in the input/output block. Both data and command chaining should not be specified in the same channel command word; if they are, data chaining takes precedence.

When a channel program includes a list of channel command words that chain data for reading operations, no channel command word may alter the contents of another channel command word in the same list. (If such alteration were allowed, specifications could be placed into a channel command word without being checked for validity. If the specifications were incorrect, the error could not be detected until the chain was completed. Data could be read into incorrect locations and the system could not correct the error.)

Control Blocks

When using EXCP, you must be familiar with the function and structure of an input/output block (IOB), an event control block (ECB), a data control block (DCB), and a data extent block (DEB). Brief descriptions of these control blocks follow. Their fields are illustrated in the section "Macro Specifications for Use with EXCP."

Input/Output Block (IOB)

The input/output block is used for communication between the problem program and the system. It provides the addresses of other control blocks, and maintains information about the channel program, such as the type of chaining and the progress of I/O operations. You must define the input/output block and specify its address as the only parameter of the EXCP macro instruction.

Event Control Block (ECB)

The event control block provides you with a completion code that describes whether the channel program was completed with or without error. A WAIT macro instruction for synchronizing I/O operations with the problem program must be directed to the event control block. You must define the event control block and specify its address in the input/output block.

Data Control Block (DCB)

The data control block provides the system with information about the characteristics and processing requirements of a data set to be read or written by the channel program. A data control block must be produced by a DCB macro instruction that includes parameters for EXCP. If appendages are not being used, a short DCB is constructed. Such a DCB does not support reduced error recovery. You specify the address of the data control block in the input/output block.

Data Extent Block (DEB)

The data extent block contains one or more extent entries for the associated data set, as well as other control information. An extent defines all or part of the physical boundaries on an I/O device occupied by, or reserved for, a particular data set. Each extent entry contains the address of a unit control block (UCB), which provides information about the type and location of an I/O device. More than one extent entry can contain the same UCB address. (Unit control blocks are set up at system generation time and need not concern you.) For all I/O devices supported by the operating system, the data extent block is produced during execution of the OPEN macro instruction for the data control block. The system places the address of the data extent block into the data control block.

Channel Program Execution

This section explains how the system uses your channel program and control blocks after you issue EXCP.

Initiation of the Channel Program

By issuing EXCP, you request the execution of the channel program specified in the input/output block. The I/O supervisor validates the request by checking certain fields of the control blocks associated with this request. If the I/O supervisor detects invalid information in a control block, it initiates abnormal termination procedures.

The I/O supervisor gets:

- the address of the data control block from the input/output block
- the address of the data extent block from the data control block
- the address of the unit control block from the data extent block

The I/O supervisor places the IOB, TCB, DEB, and UCB addresses and other information about the channel program into an area called a request queue element (RQE). The I/O supervisor uses RQEs to form logical channel queues of scheduled I/O operations. Unless you are providing appendage routines (described in the section “Appendages”), you should not be concerned with the contents of RQEs.

If you are operating in a pageable partition, the I/O supervisor now prepares to translate your virtual channel program into a real channel program. It does this by initializing translation tables and fixing the pages containing the control blocks associated with your request. If you are providing appendages, the I/O supervisor also passes control to your page fix appendage to permit you to specify the pages that must be fixed to prevent page exceptions from occurring during execution of one of your appendages. (A page exception during appendage execution causes abnormal termination.) The I/O supervisor then fixes the pages you specify upon return from your page fix appendage.

Next the I/O supervisor determines whether a channel and the requested I/O device are ready for the channel program. If they are not ready, the RQE is placed in the appropriate logical channel queue, and control is returned to the problem program. Later, when a channel and device are ready, the I/O supervisor resumes control to start the I/O operation.

If you have provided a start I/O (SIO) appendage, the I/O supervisor now passes control to it. The return address from the SIO appendage determines whether the I/O supervisor must:

- execute the I/O operation normally,
- skip the I/O operation, or
- perform extended translation on the channel program before beginning the I/O operation.

See “Appendages” in this chapter for a description of the SIO appendage and its linkage to the I/O supervisor.

If you are issuing EXCP from in a pageable partition, the channel program you construct contains virtual addresses. However, because channels cannot use virtual addresses, the I/O supervisor must:

- translate your virtual channel program into one that uses only real addresses, and
- fix in real storage the pages used as I/O areas for the data transfer operations specified in your channel program.

The I/O supervisor builds the translated (real) channel program in a portion of real storage called the System Queue Area. If the I/O device is other than a 2314 or 2319 direct-access device or a magnetic tape device, the I/O supervisor then places the address of the start of the translated channel program into the channel address word (CAW) and issues a start input/output (SIO) instruction.

Before issuing the SIO instruction for a 2314 or 2319 direct-access device, the I/O supervisor issues an initial (or stand-alone) seek, which is overlapped with other operations. You specify the seek address in the input/output block. When the seek has completed, the I/O supervisor constructs a command chain to reissue the seek, sets the file mask specified in the data extent block, and passes control to your real channel program. (When using OS/VS, you cannot issue the initial seek or set the file mask yourself. The file mask is set to prohibit Seek Cylinder commands, or, if space is allocated by tracks, Seek Track commands. If the data set is open for INPUT or RDBACK, Write commands are also prohibited.)

Before issuing SIO for a magnetic tape device, the I/O supervisor constructs a command chain to set the mode specified in the data extent block and passes control to your real channel program. (When using OS/VS, you cannot set the mode yourself.)

Modification of a Channel Program During Execution

Any user problem program that modifies an active channel program by data read in by either the I/O operation or by CPU instructions must be run in a nonpageable partition. It cannot run in a pageable partition because of the channel program translation performed by the I/O supervisor. (In a pageable partition, an attempt to modify an active channel program affects only the virtual image of the channel program, not the real channel program being executed by the channel.)

A program of this type can be changed to run in a pageable partition by either:

- executing the modified portion of the channel program as a separate I/O operation, or
- using the SIO Extend and PCI Modify appendage interfaces of the I/O supervisor, described in the section on appendages.

Completion of Execution

The system considers the channel program completed when it receives an indication of a channel end condition in the channel status word (CSW). When channel end occurs, the request element for the channel program is made available, and a completion code is placed into the event control block. The completion code indicates whether errors are associated with channel end. If device end occurs simultaneously with channel end, errors associated with device end (i.e., unit exception or unit check) are also accounted for.

If device end occurs after channel end and an error is associated with device end, the completion code in the event control block does not indicate the error. However, the status of the unit and channel is saved in the unit control block (UCB) for the device, and the UCB is marked as intercepted. The input/output block for the next request directed to the I/O device is also marked as intercepted. The error is assumed to be permanent, and the completion code in the event control block for the intercepted request indicates interception. The IFLGS field of the data control block is also flagged to indicate a permanent error. Note that when a Write Tape Mark or Erase Long Gap CCW is the last (or only) CCW in your channel program, the I/O supervisor will not attempt recovery procedures for device end errors. In these circumstances, command chaining a NOP CCW to your Write Tape Mark or Erase Long Gap CCW ensures initiation of device end error recovery procedures.

To be prepared for device end errors, you should be familiar with device characteristics that can cause such errors. After one of your channel programs has terminated, you should not release buffer space until you have determined that your next request for the device has not been intercepted. You may reissue an intercepted request.

Interruption Handling and Error Recovery Procedures

An I/O interruption allows the CPU to respond to signals from an I/O device which indicate either termination of a phase of I/O operations or external action on the device. A complete explanation of I/O interruptions is contained in *IBM System/370 Principles of Operation*, GA22-7000. For descriptions of interruptions by specific devices, refer to IBM publications for each device.

If error conditions are associated with an interruption, the I/O supervisor schedules the appropriate device-dependent error routine. The channel is then restarted with another request that is not related to the channel program in error. (The paragraphs following this one under this topic discuss "related" channel programs.) If the error recovery procedures fail to correct the error, the system places ones in the first two bit positions of the IFLGS field of the data control block. You are informed of the error by an error code that the system places into the event control block.

Related channel programs are requests that are associated with a particular data control block and data extent block in the same job step. They must be executed in a definite order, i.e., the order in which the requests are received by the I/O supervisor. A channel program is not started until all previous requests for related channel programs have been completed. You specify, in the input/output block, whether the channel program is related to others.

If a permanent error occurs in a channel program that is related to other requests, the request elements for all the related channel programs are removed from their queue and made available. This process is called purging. The addresses of the input/output blocks for the related channel programs are chained together, with the address of the

first input/output block in the chain placed into the DEBUSPRG field of the data extent block. The address of the second input/output block is placed into the IOBRESTR field of the first input/output block, and so on. The last input/output block in the chain is indicated by all ones in the last byte of the IOBRESTR field. The chain defines the order in which the request elements for the related channel programs are removed from the request queue.

For all requests related to the channel program in error, the system places completion codes into the event control blocks. The DCBIFLGS field of the data control block is also flagged. Any requests for a data control block with error flags are posted complete without execution. To reissue requests related to the channel program in error, you must reset the first two bits of the DCBIFLGS field of the data control block to zeros. You then issue a RESTORE macro instruction, specifying, as the only parameter, the address of the DEBUSPRG field of the data extent block. This causes execution of all the related channel programs. (The RESTORE macro definition and how to add it to the macro library are in “System macro Instructions.”) Alternatively, to restart only particular channel programs rather than all of them, you may reissue EXCP for each channel program desired.

Appendages

This section discusses the appendages that you may optionally code when using EXCP. Before a programmer-written appendage can be executed, it must be included in the SVC library. These procedures are explained first; descriptions of the routines themselves and of their coding specifications follow.

An appendage must be a member of the SVC library. The full member name of an appendage is eight bytes in length, but the first six bytes are required by IBM standards to be the characters IGG019. The last two characters must be provided by you as an identification; they may range in collating sequence from WA to Z9.

The SVC library is a partitioned data set named SYS1.SVCLIB. If you are using VS1, you can insert an appendage into the SVC library during the system generation process or by link-editing it into the SYS1.SVCLIB. If you are using VS2, you can insert an appendage into SYS1.LPALIB when you generate your system, or you can link-edit the appendage into SYS1.LPALIB.

To enter a routine into the SVC library during system generation, use the SVCLIB macro instruction, which is described in *OS/VS1 System Generation Reference*, GC26-3791.

To enter an appendage into OS/VS2 during system generation, use the LPALIB macro, which is described in *OS/VS2 System Generation Reference*, GC26-3792.

An appendage is a programmer-written routine that provides additional control over I/O operations. By providing appendages, you can examine the status of I/O operations and determine the actions to be taken for various conditions. An appendage may receive control when one of the following occurs:

- Page fixing
- Start I/O
- Program controlled interruption
- End of extent
- Channel end
- Abnormal end

An appendage is executed in supervisor state. An appendage must not issue any SVC instructions or instructions that change the status of the computing or operating system (for example, WTO, LPSW, or any privileged instructions). Because an appendage runs disabled for all types of interrupts except for machine checks, it must not enter loops that test for completion of I/O operations. An appendage must not alter storage that is used by either the supervisor or the I/O supervisor.

The last two characters of an appendage's 8-character name must be specified in the DCB macro instruction, as described in the section "Macro Specification for Use with EXCP". When an OPEN macro instruction for the data control block is issued, any appendages specified in the DCB macro instruction are loaded into the problem program partition. They are loaded into virtual storage if the partition is pageable.

Your appendage routines are made available to the I/O supervisor when the DCB for the data set is opened. The address of each appendage you have provided (and the number of 2K segments of storage each occupies) is placed in a table called the appendage vector table. This table is always constructed by the system when OPEN is issued; if an appendage is not provided, the table contains the address of a branch (BR 14) instruction that immediately returns control to the I/O supervisor. Using the appendage vector table, the I/O supervisor branches and links to each appendage at the appropriate time.

The I/O supervisor uses registers to pass parameters to the appendages as follows:

- *Register 1:* Address of the request queue element (RQE) for the channel program. The RQE consists of 20 bytes formatted as shown in Figure 10.
- *Register 2:* Address of the input/output block (IOB).
- *Register 3:* Address of the data extent block (DEB).
- *Register 4:* Address of the data control block (DCB).
- *Register 7:* Address of the unit control block (UCB).
- *Register 14:* Address of the location in the I/O supervisor to which control is to be returned after execution of the appendage. When returning control to the I/O supervisor, you may use displacements from the return address in register 14. Allowable displacements are summarized in the following table and described later for each appendage.
- *Register 15:* Address of the entry point of the appendage, except in the instance of the page fix appendage. Refer to the following table.

You may not change register 1 in an appendage; this is reserved in case an abnormal condition occurs while the appendage is in control. Register 9, if used, must be set to binary zero before control is returned to the system. All other registers, except those indicated in the descriptions of each appendage, must be saved and restored if they are used. Figure 11 summarizes register conventions.

The types of appendages are listed in the following paragraphs, with explanations of when they are entered, how they return control to the system, and which registers they may use without saving and restoring.

0(0)	TSTLNK Address of Next RQE in This Queue	2(2)	TSTUCB Address of UCB
4(4)	TSTIDT TCB Identification	5(5)	TSTIOB Address of IOB
8(8)	TSTPRI Requestor's Priority	9(9)	TSTDEB Address of DEB
12(0C)	TSTKEY Requestor's Protection Key	13(0D)	TSTTCB Address of TCB
16(10)	Channel Program Translation Flags	17(11)	Address of Channel Program Translation Header Block

Figure 10. The Request Queue Element (RQE)

Appendages	Entry Point	Returns	Available Work Reg.*
EOE	Reg 15	Reg 14 + 0 Reg 14 + 4 Reg 14 + 8	Extent Error Return Skip Try Again Reg. 10, 11, 12, and 13
SIO	Reg 15	Reg 14 + 0 Reg 14 + 4 Reg 14 + 8	Normal Skip Extend Reg. 10, 11, and 13
PCI	Reg 15	Reg 14 + 0 Reg 14 + 4	Normal Modify Reg. 10, 11, 12, and 13
CE	Reg 15	Reg 14 + 0 Reg 14 + 4 Reg 14 + 8 Reg 14 + 12	Normal Skip Re-EXCP By-Pass Reg. 10, 11, 12, and 13
XCE	Reg 15	Reg 14 + 0 Reg 14 + 4 Reg 14 + 8 Reg 14 + 12	Normal Skip Re-EXCP By-Pass Reg. 10, 11, 12, and 13
PGFX	Reg 15 + 4	Reg 14 + 0 Reg 14 + 4	Normal fix list Extended fix list Reg. 10, 11, and 13

*Certain register conventions for passing parameters from appendages to the I/O supervisor must be followed. These conventions are described in the appendage descriptions.

Figure 11. Entry Points, Returns, and Available Work Registers for the I/O Supervisor Appendages

Page Fix (PGFX) and Start I/O (SIO) Appendage

This appendage comprises two essentially independent appendages. The total appendage can be viewed as a re-enterable subroutine having two entry points, one for SIO and one for PGFX.

The SIO entry point is offset 0 in the subroutine; it may be a branch to another area of the appendage. The entry point to the PGFX appendage is at offset +4 in the subroutine.

Page Fix (PGFX) Appendage

The purpose of this appendage is to list all of the areas that must be fixed (made nonpageable) to prevent paging exceptions from occurring during the execution of appendages related to the current I/O request. This appendage may be entered more than once. However, each time it is entered, it must create the same list of areas to be fixed, including the boundary of any items used to create the list. After the first entry to this appendage, any paging exceptions occurring during processing of this or related appendages cause abnormal termination.

Normal Page Fix List Processing

On entry to this appendage, register 10 points to a work area for a list of 7 page fix entries of 8 bytes each. Each page fix entry placed in the list by the appendage must have the following doubleword format:

X'00'	Starting Virtual Address of Area to be Fixed	X'00'	Ending Virtual Address of Area to be Fixed + 1
← 1 Byte →	← 3 Bytes →	← 1 Byte →	← 3 Bytes →

On return to the I/O supervisor (via the return address provided in register 14), register 10 must point to the first page fix entry in the work area, and register 11 must contain the number of page fix entries in the work area. The I/O supervisor then fixes the pages corresponding to the areas listed by the PGFX appendage. The pages remain fixed until the associated I/O request queue element is dequeued from its logical channel queue.

Extended (10-entry) Page Fix List Processing

For installations using VS2, as many as 10 entries (three more than allowed for the normal list) are allowed. At entry to the page-fix appendage, register 10 points to an area to be used for a page-fix list. If your DCB, IOB, or ECB (DECB) exceed the sizes fixed by the I/O supervisor, you must include one or more entries in the fix list that contain these control blocks. The sizes of the control blocks fixed by I/O supervisor follow:

DCB	104 bytes*
IOB	80 bytes + a 16-byte prefix, for a total of 96 bytes.
ECB (DECB)	32 bytes

*In installations using VS1, Release 1, the I/O supervisor fixes a 252-byte area for the DCB.

You must return to I/O supervisor using a branch entry to the location at register 14 + 4. Register 10 must point to the extended fix list and register 11 must contain the number of entries in the extended fix list. The I/O supervisor will then fix the pages containing the entries in your fix list. These pages remain fixed until the associated request queue element is dequeued from its logical channel queue.

To use extended channel program translation, you must provide the address of the first entry in parameter list in register 10 and put the count of the number of entries in the list in register 11. In addition, you must provide a count of the number of CCWs in the list or a count of the number of indirect addresses associated with the CCW. You must set bit 0 of the count field (called the I bit) to 0 to indicate that bits 1-7 contain a count of the contiguous virtual CCWs; set the I bit to 1 to indicate that bits 1-7 contain a count of the indirect addresses associated with the CCW.

Start I/O (SIO) Appendage

Unless an error procedure is in control, the I/O supervisor passes control to the SIO appendage just before the I/O supervisor translates your channel program. You have an opportunity to modify or extend his channel program after he requests the I/O operation. However, you should not alter the IOBSTART field of your IOB in the SIO appendage; changing the address in IOBSTART has no effect on where the I/O supervisor begins CCW translation. If the I/O activity is not initiated because of a busy condition and the I/O request has not been translated, this appendage is re-entered before the SIO instruction is issued; otherwise, it is not re-entered.

Optional return vectors give the I/O requestor the following choices:

Reg. 14 + 0

Normal return. Normal channel program translation and SIO instruction execution occur.

Reg. 14 + 4

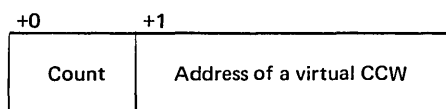
Skip the I/O operation. The channel program is not posted complete, but the request queue element is made available. The I/O requestor may post the channel program as follows:

1. Save necessary registers.
2. Place pointer to post entry address from the communications vector table (CVT) in register 15.
3. Place current TCB address from the CVT in register 12.
4. Place ECB address from the IOB in register 11.
5. Set the completion code in the high-order byte in register 10.
6. Go to POST using BALR 14,15.

Reg. 14 + 8

Extended channel program translation. The SIO appendage must define to the I/O supervisor *all* of the channel command words (CCWs) that *can* be used in the forthcoming I/O operation. The definition includes the CCW chain pointed to by the IOBSTART field of the input/output block, the normal starting CCW location. Through this definition the SIO appendage indicates to the I/O supervisor the CCWs that can be changed in a program controlled interrupt (PCI) appendage. The definition thus enables the I/O supervisor to translate much of the requestor's channel program before the SIO instruction is issued. In this way the amount of translation performed following a PCI appendage is minimized. To use extended channel program translation, you must provide the address of the first CCW in register 10 and put the count of the number of CCWs in the list in register 11. In

addition, you must provide a count of the number of CCWs in the list or a count of the number of indirect addresses associated with the CCW. You must set bit 0 of the count field (called the I bit) to 0 to indicate that bits 1-7 contain a count of the contiguous virtual CCWs; set the I bit to 1 to indicate that bits 1-7 contain a count of the indirect addresses associated with the CCW.



An I = 0 entry must precede an I = 1 entry.

Program Controlled Interruption (PCI) Appendage

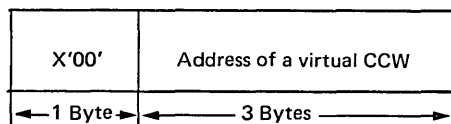
This appendage is entered when a program controlled interruption occurs. At the time of the interruption, the contents of the channel status word will not have been placed in the “channel status word” field of the input/output block. The channel status word can be obtained from location 64.

You may use registers 10 through 13 in a PCI appendage without saving and restoring their contents. This appendage may be reentered for the same channel program if the error recovery procedure is in the process of retrying a CCW with the program controlled interrupt (PCI) bit set on. The IOB error flag is set when the error recovery procedure is in control (IOBFLAG1 = X'20').

Refer to the topic “Block Multiplexor Channel Programming Notes” later in this chapter for special PCI conditions encountered with command retry.

To return control to the I/O supervisor for normal interruption processing, use the return address in register 14. To make use of the “PCI Modify” interface of the I/O supervisor, use register 14 + 4 as the return address.

The PCI Modify interface enables you to make changes to translated (real) CCWs in the midst of an I/O operation. In the PCI appendages you can make changes to the virtual image of the channel program whose execution is interrupted. To cause the I/O supervisor to make corresponding changes to the real channel program, you must construct a “PCI Modify Parameter List” in a fixed area of storage that you provide. The list must contain the virtual address of each CCW changed. Each entry in the list consists of four bytes as follows:



On exit from the PCI appendage (via register 14 + 4), register 10 must point to the first entry in the list, and register 11 must contain the number of entries in the list.

The I/O supervisor then finds the real CCW corresponding to each virtual CCW specified in the list, translates the virtual CCW to real, and replaces the real CCW. If the CCW requires entries from the indirect address list, an entry is provided.

Transfer-in-channel (TIC) commands are resolved to previously defined CCW strings only, and cannot be used to expose new CCW strings. Also, new pages to be fixed cannot be exposed now.

Error conditions created by incorrect specifications of PCI Modify Parameter List entries abnormally terminate the I/O requestor. Examples of such error conditions are:

- The virtual CCW listed exposes a new CCW string or data page.
- A page exception is encountered in accessing an entry in the list.

End-of-Extent Appendage

This appendage is entered when the seek address specified in the input/output block is outside the allocated extent limits indicated in the data extent block.

If you use the return address in register 14 to return control to the system, the abnormal end appendage is entered. An end-of-extent error code (X'42') is placed in the "ECB code" field of the input/output block for subsequent posting in the ECB.

You may use the following optional return addresses:

- Contents of register 14 plus 4 – The channel program is posted complete; its request element is returned to the available queue.
- Contents of register 14 plus 8 – The request is tried again.

You may use registers 10 through 13 in an end-of-extent appendage without saving and restoring their contents.

Note: If an end-of-cylinder or file-protect condition occurs, the I/O supervisor updates the seek address to the next higher cylinder or track address, and re-executes the request. If the new seek address is within the data set's extent, the request is executed; if the new seek address is not within the data set's extent, the end-of-extent appendage is entered. If you wish to try the request in the next extent, you must move the new seek address into the UCB at UCB+48.

If a file protect is caused by a full seek (command code=07) embedded within a channel program, the request is flagged as a permanent error, and the abnormal end appendage is entered.

Channel End (CE) Appendage

This appendage is entered when a channel end (CE), unit exception (VEX) with or without channel end, or channel end with wrong length record (WLR) occurs without any other abnormal end conditions.

If you use the return address in register 14 to return control to the I/O supervisor, the channel program is posted complete, and its request element is made available. In the case of unit exception or wrong length record, the error recovery procedure is performed before the channel program is posted complete, and the IOBEX flag (X'04') in IOBFLAG1 is set on. The condition code may be directly tested by using a BC

instruction. A CC=0 means no UEX or WLR accompanied this interruption. The CSW status may be obtained from the IOBCSW field.

If the appendage takes care of the wrong length record and/or unit exception, it may turn off the IOBEX (X'04') flag in IOBFLAG1 and return normally. The event will then be posted complete (completion code X'7F' under normal conditions, taken from the high-order byte of the IOBECBCC field). If the appendage returns normally without resetting the IOBEX flag to zero, the request will be routed to the associated device error routine, and then the abnormal end appendage will be immediately entered with IOBECBCC completion code is set to X'41'.

You may use the following optional return addresses:

- Contents of register 14 plus 4 – The channel program is not posted complete, but its request element is made available. You may post the event by using the calling sequence described under the start I/O appendage. This is especially useful if you wish to post an ECB other than the IOBECB.
- Contents of register 14 plus 8 – The channel program is not posted complete, and its request element is placed back on the request queue so that the I/O operation can be retried. For correct re-execution of the channel program, you must re-initialize the IOBFLAG1, IOBFLAG2, and IOBFLAG3 fields of the input/output block and set the “Error Counts” field to zero. As an added precaution, the IOBSENS0, IOBSENS1, and IOBCSW fields should be cleared.
- Contents of register 14 plus 12 – The channel program is not posted complete, and its request element is not made available. (The request element is assumed to be used in a subsequent asynchronous exit routine.)

You may use registers 10 through 13 in a channel end appendage without saving and restoring their contents.

Abnormal End (XCE) Appendage

This appendage may be entered on abnormal conditions, such as: unit check, unit exception, wrong length indication, program check, protection check, channel data check, channel control check, interface control check, chaining check, out-of-extent error, and intercept condition (i.e., device end error). It may also be entered when an EXCP is issued for a DCB that has already been purged.

1. When this appendage is entered due to a unit exception and/or wrong length record indication, the IOBECBCC is set to X'41'. For further information on these conditions see “Channel End Appendage.”
2. When the appendage is entered due to an out-of-extent error, the IOBECBCC is set to X'42'.
3. When the appendage is first entered due to an intercept condition, the IOBECBCC is set to X'7E'. If it is then determined that the error condition is permanent, the appendage will be entered a second time with the IOBECBCC set to X'44'. The intercept condition signals that an error was detected at device end after channel end on the previous request.
4. When the appendage is entered due to an EXCP being issued to an already purged DCB, this request will enter the abnormal end appendage with the IOBECBCC set to X'48'. This applies only to related requests.

5. When the appendage is entered with the IOBECBCC set to X'7F', it may be due to a unit check, program check, protection check, channel data check, channel control check, interface control check, or chaining check. When the IOBECBCC is X'7F', it may be the first detection of an error in the associated channel program, or it could occur after an error routine has attempted to correct the error but was unsuccessful in its retry. Under these two conditions, the IOB error flag is set in IOBFLAG1; it indicates that the error routine has declared the error to be permanent. When the IOBEX flag (bit 5 of the IOBFLAG1) is on, the IOBECBCC field will contain a 41, 42, 48, 4B, or 4F in hexadecimal, indicating a permanent I/O error.

To determine if an error is permanent, you should check the IOBECBCC field of the IOB. To determine the type of error, check the channel status word and the sense information in the IOB. However, when the IOBECBCC is X'42' or X'48', these fields are not applicable. For X'44' the CSW is applicable, but the sense is valid only if the unit check bit is set. If you use the return address in register 14 to return control to the system, the channel program is posted complete, and its request element is made available. (The SYNADAF macro instruction, described in *OS/VS Data Management Macro Instructions*, may be used in an error analysis routine to analyze permanent I/O errors.) You may use the following optional return addresses:

- Contents of register 14 plus 4 – The channel program is not posted complete, but its request element is made available.
- Contents of register 14 plus 8 – The channel program is not posted complete, and its request element is placed back on the request queue so that the request can be retried. For correct re-execution of the channel program, you must re-initialize the IOBFLAG1, IOBFLAG2, and IOBFLAG3 fields of the input/output block and set the IOBERRCT field to zero. As an added precaution, the IOBSENS0, IOBSENS1, and IOBCSW fields should be cleared.
- Contents of register 14 plus 12 – The channel program is not posted complete, and its request element is not made available. (The request element is assumed to be used in a subsequent asynchronous exit.)

You may use registers 10 through 13 in an abnormal end appendage without saving and restoring their contents.

Block Multiplexer Channel Programming Notes

Command retry is a function of the block multiplexer channel supporting the 3330 Disk Storage and the 2305 Fixed Head Storage devices. When the channel receives a retry request, it repeats the execution of the channel command word (CCW) requiring no additional input/output interrupts. For example, a control unit may initiate a retry procedure to recover from a transient error.

A command retry during the execution of a channel program may cause any of the following conditions to be detected by the initiating program:

- *Modifying CCWs:* A CCW used in a channel program must not be modified before the CCW operation has been successfully completed. Without the command retry function, a command was fetched only once from storage by a channel. Therefore, a program could determine through condition codes or program controlled interruptions (PCI) that a CCW had been fetched and accepted by the channel. This permitted the CCW to be modified before re-execution. With the command retry function, this cannot be done, since the channel will fetch the CCW from storage again on a command retry sequence. In the case of data chaining, the channel will command retry starting with the first CCW in the data chain.
- *Program Controlled Interrupts:* A CCW containing a PCI flag may cause multiple program controlled interruptions to occur. This happens if the PCI-flagged CCW was retried during a command retry procedure, and a PCI could be generated each time the CCW is re-executed.
- *Residual Count:* If a channel program is prematurely terminated during the retry of a command, the residual count in the channel status word (CSW) will not necessarily indicate how much storage was used. For example, if the control unit detects a “wrong length record” error condition, an erroneous residual count is stored in the CSW until the command retry is successful. When the retry is successful, the residual in the CSW is the correct length of the data transfer. Since the channel will not allow more data to be transferred than is specified in the count field of the CCW, this situation will occur only when reading variable records or unknown record types.
- *Command Address:* When data chaining with command retry, the CSW may not indicate how many CCWs have been executed at the time of a PCI. For example:

CCW#	Channel Program
1	Read data chain
2	Read data chain
3	Read data chain, PCI
4	Read command chain

In this example, assume that the control unit signals command retry on Read #3 and the CPU accepts the PCI after the channel resets the command address to Read #1 because of command retry. The CSW stored for the PCI will contain the command address of Read #1, when actually the channel has progressed to Read #3.

- *“Bit Spinning” on Data Read:* Any program that tests a data storage location to determine when a CCW has been executed and continues to execute based on this data may get incorrect results if an error is detected and the CCW is retried. An example of this is a PCI appendage in which ones are placed in a buffer area that will be overlaid with zeros when a record is read. When the PCI appendage is entered, a check for zeros is made and the appendage will continue to loop until the record is read into the buffer (indicated by ones changed to zeros). If the appendage uses the data from this record to modify a channel program, the results will be unpredictable during a command retry sequence, as the CCW has not been correctly executed.

Macro Specifications for Use With EXCP

If you are using the EXCP macro instruction, you must also use DCB, OPEN, CLOSE, and, in some cases, the EOVS macro instruction. The parameters of these macro instructions and the EXCP macro instructions are explained here. A diagram of the data control block is included with the description of the DCB macro instruction.

DCB – Define Data Control Block for EXCP

The EXCP form of the DCB macro instruction produces a data control block that can be used with the EXCP macro instruction. You must issue a DCB macro instruction for each data set to be processed by your channel programs. Notation conventions and format illustrations of the DCB macro instruction are given in the *Data Management Macro Instructions* publication. DCB parameters that apply to EXCP may be divided into four categories, depending on the following portions of the data control block that are generated when they are specified:

- *Foundation block.* This portion is required and is always 12 bytes in length. You must specify two of the parameters in this category.
- *EXCP interface.* This portion is optional. If you specify any parameter in this category, 20 bytes are generated.
- *Foundation block extension and common interface.* This portion is optional and is always 20 bytes in length. If this portion is generated, the device dependent portion is also generated.
- *Device dependent.* This portion is optional and is generated only if the foundation block extension and common interface portion is generated. Its size ranges from 4 to 20 bytes, depending on specifications in the DEVD parameter. However, if you do not specify the DEVD parameter (and the foundation extension and common interface portion is generated), the maximum 20 bytes for this portion are generated.

Some of the procedures performed by the system when the data control block is opened and closed (such as writing file marks for output data sets on direct access volumes) require information from optional data control block fields. You should make sure that the data control block is large enough to provide all information necessary for the procedures you want the system to handle.

Figure 12 shows the relative position of each portion of an opened data control block. The fields corresponding to each parameter of the DCB macro instruction are also designated, with the exception of DDNAME, which is not included in a data control block that has been opened. The fields identified in parentheses represent system information that is not associated with parameters of the DCB macro instruction.

Sources of information for data control block fields other than the DCB macro instruction are data definition (DD) statements, data set labels, and data control block modification routines. You may use any of these sources to specify DCB parameters. However, if a portion of the data control block is not generated by the DCB macro instruction, the system does not accept information intended for that portion from any alternative source.

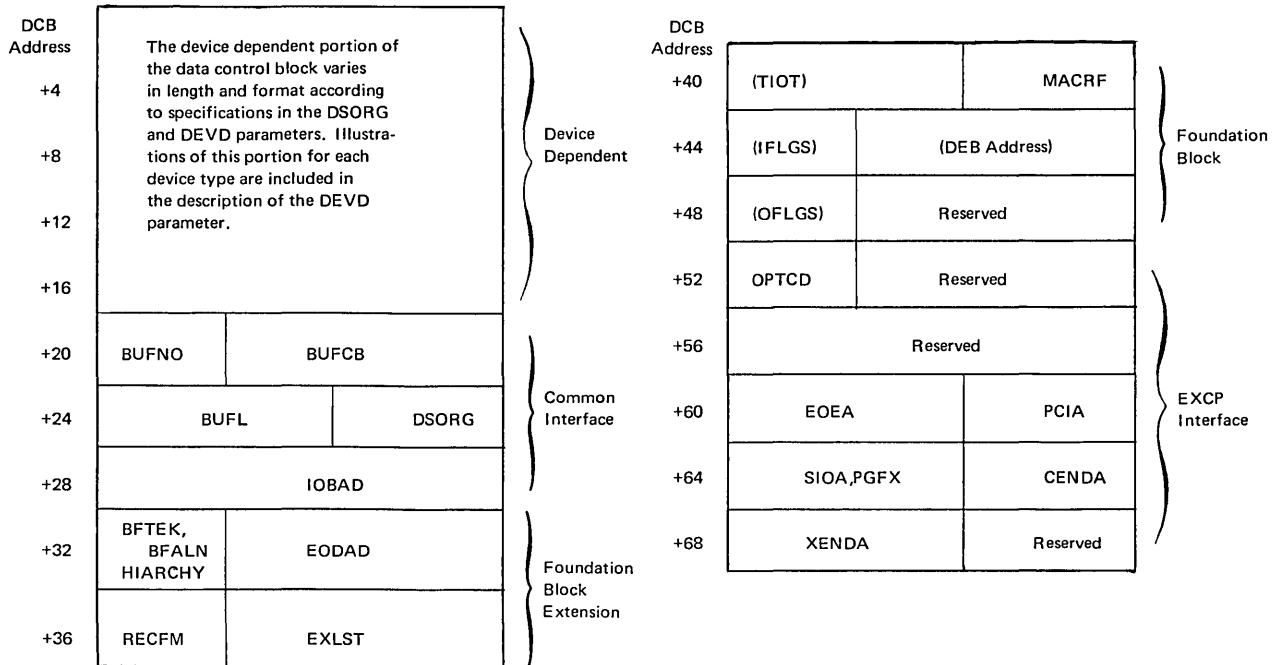


Figure 12. Data Control Block Format for EXCP (After OPEN)

Foundation Block Parameters

DDNAME=symbol

The name of the data definition (DD) statement that describes the data set to be processed. This parameter must be given.

MACRF=(E)

The EXCP macro instruction is to be used in processing the data set. This operand must be coded. If, however, you are processing a multivolume, direct data set (DSORG=DA), you should code MACRF=(W) or (R). This results in the operating system's performing automatic volume mounting and causes the Open routines to complete the data extent block (DEB) that represents all volumes of the data set. Do not use MACRF=(W) or (R) if you want to use your own appendages; they will not be loaded.

REPOS= Y

N

Magnetic tape volumes: If your system generation statements include the dynamic device reconfiguration (DDR) entry, then this parameter controls whether the DDR routine will attempt to reposition the volume after swapping devices. (To have the

DDR routine attempt to reposition your tape volume, you must maintain the block count in the DCBBLKCT field.)

Y – Yes, attempt to reposition.

N – No, do not attempt to reposition.

If the operand is omitted, N is assumed.

EXCP Interface Parameters

EOEA=symbol

2-byte identification of an end-of-extent appendage that you have entered into the SVC or LPA library. (See Note A.)

PCIA=symbol

2-byte identification of a program controlled interruption (PCI) appendage that you have entered into the SVC or LPA library. (See Note A.)

SIOA=symbol

2-byte identification of a start I/O (SIO) appendage that you have entered into the SVC or LPA library. (See Note A.)

CENDA=symbol

2-byte identification of a channel end appendage that you have entered into the SVC or LPA library. (See Note A.)

XENDA=symbol

2-byte identification of an abnormal end appendage that you have entered into the SVC or LPA library. (See Note A.)

Note A: The full name of an appendage is 8 bytes in length, but the first six bytes are required by IBM standards to be the characters IGG019. You provide the last two characters as the identification; they may range in collating sequence from WA to Z9.

PGFX= {Yes|No}

A yes response indicates the existence of a user page fix appendage. If you specify PGFX=yes, also specify SIOA=XX.

OPTCD=Z

indicates that for magnetic tape (input only) a reduced error recovery procedure (5 reads only) will occur when a data check is encountered. It should be specified only when the tape is known to contain errors and the application does not require that all records be processed. Its proper use would include error frequency analysis in the SYNAD routine. Specification of this parameter will also cause generation of a foundation block extension. This parameter is ignored unless it was selected at system generation.

IMSK=value

Any specification indicates that the system will not use IBM-supplied error routines.

Foundation Block Extension and Common Interface Parameters

EXLST=address

the address of an exit list that you have written for exceptional conditions. The format of this exit list is given in *OS/VS Data Management Services Guide*.

EODAD=address

the address of your end-of-data set routine for input data sets. If this routine is not available when it is required, the task is abnormally terminated.

DSORG= {PS}
 {PO}
 {DA}
 {IS}

the data set organization (one of the following codes). Each code indicates that the format of the device-dependent portion of the data control block is to be similar to that generated for a particular access method:

Code	DCB Format for
PS	QSAM or BSAM
PO	BPAM
DA	BDAM
IS	QISAM or BISAM

Note: For direct-access devices, if you specify either PS or PO, you must maintain the following fields of the device-dependent portion of the data control block so that the system can write a file mark for output data sets:

- The track balance (DCBTRBAL) field, which contains a 2-byte binary number that indicates the remaining number of bytes on the current track.
- The full disk address (DCBFDAD) field, which indicates the location of the current record. The address is in the form MBBCCHHR.

IOBAD=address

the address of an input/output block (IOB). If a pointer to the current IOB is not required, you may use this field for any purpose.

The following parameters are not used by the EXCP routines, but they provide cataloging information about the data set. This information can be used later by access method routines that read or update the data set.

RECFM=code

the record format of the data set. Record format codes are given in *OS/VS Data Management Macro Instructions*. When writing a data set to be read later using one of the access method routines, the RECFM, LRECL, and BLKSIZE should be specified to identify the data set attributes. LRECL and BLKSIZE can only be specified in a DD statement, since these fields do not exist in a DCB used by EXCP.

BFTEK={S | E}

the buffer technique, either simple or exchange.

BFALN={F | D}

the word boundary alignment of each buffer, either fullword or doubleword.

BUFL=length

the length in bytes of each buffer; the maximum length is 32,767.

BUFNO=number

the number of buffers assigned to the associated data set; the maximum number is 255.

BUFCB=address

the address of a buffer pool control block, i.e., the 8-byte field preceding the buffers in a buffer pool.

Device-Dependent Parameters

DEVD=code

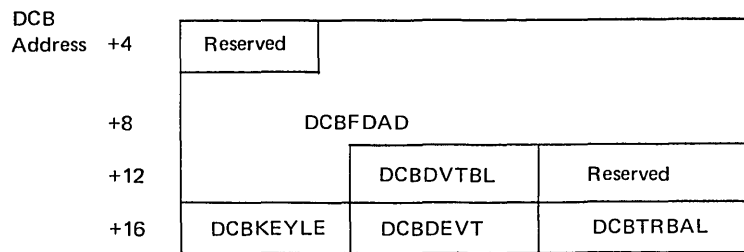
the device on which the data set may reside. The codes are listed in order of descending space requirements for the data control block:

Code	Device
DA	Direct access
TA	Magnetic tape
PT	Paper tape
PR	Printer
PC	Card punch
RD	Card reader

Note: If you do not wish to select a specific device until job set-up time, you should specify the device type requiring the largest area.

The following diagrams illustrate the device-dependent portion of the data control block for each device type specified in the DEVD parameter, and for each data set organization specified in the DSORG parameter. Fields that correspond to device-dependent parameters in addition to DEVD are indicated by the parameter name. For special services, you may have to maintain the fields shown in parentheses. The special services are explained in the note that follows the diagram.

Device-dependent portion of data control block when DEVD=DA and DSORG=PS or PO:

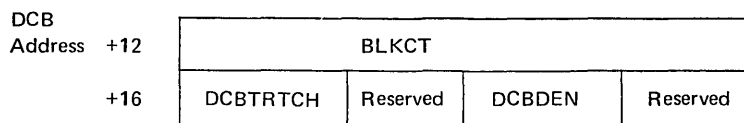


Note: For output data sets, the system uses the contents of the full disk address (DCBFDAD) field plus one to write a file mark when the data control block is closed, provided the track balance (DCBTRBAL) field indicates that space is available. You must maintain the contents of these two fields yourself if the system is to write a file mark. OPEN will initialize DCBDVTBL and DCBDEVT.

Device-dependent portion of data control block when DEVD=DA and DSORG=IS or DA:



Device-dependent portion of data control block when DEVD=TA and DSORG=PS:



Note: For output data sets, the system control program uses the contents of the block count (DCBBLKCT) field to write the block count in trailer labels when the data control block is closed or when the EOVS macro instruction is issued. You must

maintain the contents of this field yourself if the system is to write the correct block count.

When using EXCP to process a tape data set open at a checkpoint, you must be careful to maintain the correct count; otherwise, the system may position the data set incorrectly when restart occurs.

If your system generation statements include the dynamic device reconfiguration entry, this field must be maintained by you for repositioning. Also, your DCB macro instruction must include the REPOS=Y entry.

Device-dependent portion of data control block when DEVD=PT and DSORG=PS:

DCB Address +16	DCBCODE	Reserved
--------------------	---------	----------

Device-dependent portion of data control block when DEVD=PR and DSORG=PS:

DCB Address +16	DCBPRTSP	Reserved
--------------------	----------	----------

Device-dependent portion of data control block when DEVD=PC or RD and DSORG=PS:

DCB Address +16	DCBMODE,DCBSTACK	Reserved
--------------------	------------------	----------

The following DCB operands pertain to specific devices and may be specified only when the DEVD parameter is specified.

KEYLEN=length

for direct-access devices, the length in bytes of the key of a physical record, with a maximum value of 255. When a block is read or written, the number of bytes transmitted is the key length plus the record length.

CODE=value

for paper tape, the code in which records are punched:

Value	Code
I	IBM BCD
F	Friden
B	Burroughs
C	National Cash Register
A	ASCII
T	Teletype ¹
N	no conversion (format-F records only)

If this parameter is omitted, N is assumed.

DEN=value

for magnetic tape, the tape recording density in bits per inch:

Value	Density	
	Model 2400/3400 7-track	Model 2400/3400 9-track
0	200	—
1	556	—
2	800	800
3	—	1600

If this parameter is omitted, the lowest density is assumed.

¹ Trademark of Teletype Corporation

TRTCH=value

for 7-track magnetic tape, the tape recording technique:

Value Tape Recording Technique

- | | |
|---|-----------------------------------------------------------|
| C | Data conversion feature is available. |
| E | Even parity is used. (If omitted, odd parity is assumed.) |
| T | BCDIC to EBCDIC translation is required. |

MODE=value

for a card reader or punch, the mode of operation. Either C (column binary mode) or E (EBCDIC code) may be specified.

STACK=value

for a card punch or card reader, the stacker bin to receive cards, either 1 or 2.

PRTSP=value

for a printer, the line spacing, either 0, 1, 2, or 3.

OPEN – Initialize Data Control Block

The OPEN macro instruction initializes one or more data control blocks so that their associated data sets can be processed. You must issue OPEN for all data control blocks that are to be used by your channel programs. (A dummy data set may not be opened for EXCP.) Some of the procedures performed when OPEN is executed are:

- Construction of data extent block (DEB).
- Transfer of information from DD statements and data set labels to DCB.
- Verification or creation of standard labels.
- Tape positioning.
- Loading of your appendage routines.

The three parameters of the OPEN macro instruction are:

[symbol] OPEN (dcb address,[options],...)

dcb address – A-type Address or (2-12)

the address of the data control block to be initialized. (More than one data control block may be specified.)

option₁

the intended method of I/O processing of the data set. You may specify this parameter as either INPUT, RDBACK, or OUTPUT. For each of these, label processing when OPEN is executed is as follows:

- | | |
|--------|--------------------------------|
| INPUT | – Header labels are verified. |
| RDBACK | – Trailer labels are verified. |
| OUTPUT | – Header labels are created. |

If this parameter is omitted, INPUT is assumed.

option₂

the volume disposition that is to be provided when volume switching occurs. The operand values and meanings are as follows:

- | | |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| REREAD | - Reposition the volume to process the data set again. |
| LEAVE | - No additional positioning is performed at end-of-volume processing. |
| DISP | - The disposition indicated on the DD statement is tested and appropriate positioning provided. This service is assumed if this operand is omitted and volume positioning is applicable. If there is no disposition specified in the DD statement when this operand is specified, LEAVE is assumed. |

When you code MACRF=(E) in the DCB macro instruction, indicating that your program uses the EXCP macro instruction, the Open routines process your data control block as if it represents a single-volume, physical-sequential data set, except that the DCB fields are merged (from and to the DSCB and JFCB for the data set) according to the DSORG you specify in the DCB macro instruction.

However, if you are concatenating partitioned data sets, mount messages will be issued, volume verification will be performed, and a DEB will be built that represents all the extents and volumes of the concatenated data set.

You should recognize that if you are opening multiple-volume direct or index-sequential data sets, only the first volume of the data set is guaranteed to be mounted by the operating system, and the DEB built by the Open routines for the data set will represent only the first volume.

The list and execute forms of the OPEN macro instruction are described in *OS/VS Data Management Macro Instructions*.

EXCP – Execute Channel Program

The EXCP macro instruction requests the initiation of the I/O operations of a channel program. You must issue EXCP whenever you want to execute one of your channel programs. The only parameter of the EXCP macro instruction is:

[symbol] EXCP iob-address

iob-address – A-type address, (2-12), or (1)

the address of the input/output block of the channel program to be executed.

ATLAS – Assigning an Alternate Track and Copying Data from the Defective Track

A program that uses the EXCP macro instruction for input and output may use the ATLAS macro instruction, during the execution of the program, to obtain an alternate track and to copy a defective track onto the alternate track. With the use of ATLAS, the program can recover from permanent (hard) errors encountered in the execution of the following types of I/O commands:

- Search ID.
- Write. (The error condition must be confirmed during the execution of the channel program by a CCW that checks the data written.)
- Read count. Errors in the CCHHR part of the count area can be recovered from unless the record is the home address or record zero. Errors in the KDD part of the count area cannot be recovered from unless the user has identified the defective record.

Your DCB must include the DCBRECFCM field and the field must show whether the data set is in the track overflow format. If it is, recovery from errors in last records on tracks depends on your identifying the track overflow record segments.

Recovery takes the form of obtaining an alternate good track and copying the defective track onto the good alternate one. Unless a re-execution of the channel program by ATLAS can correct the defect, the user should examine, and if necessary replace, defective records in a subsequent job if the data set is to be processed again.

The format is:

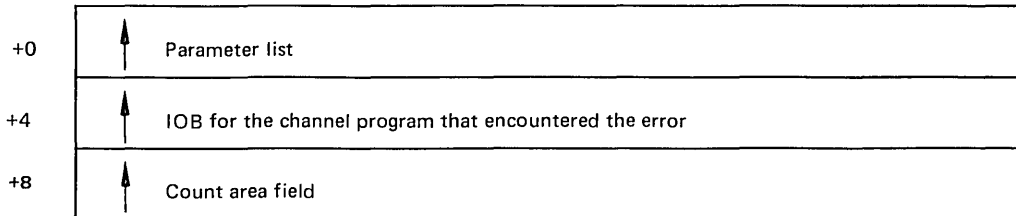
```

[symbol]      ATLAS      PARMADR = {address}[,CHANPRG = {R}]
                                     [CNTPTR = {P}] [WRITS = {YES}]
                                     [CNTPTR = {F}] [WRITS = {NO}]

```

PARMADR

Address of a parameter address list of the following format:



The count area field contains the CCHHRKDD of a defective record or the CCHH of a track that is to be copied.

address, (2-12) or (1) Address is given as the symbolic label of the address list or as the number of a general register that contains the address of the list.

{**R**}

CHANPRG= {**NR**}

specifies whether the channel program that encountered the error can be executed again.

R - Channel program may be executed again by ATLAS. Before permitting re-execution of the channel program by ATLAS, you must reset the error indications of the previous execution fields in the DCBIFLGS. (See the example of the use of ATLAS below.)

NR - Channel program may not be executed again.

If this parameter is omitted, R is assumed.

CNTPTR

specifies whether the count area field contains a full count area (CCHHRKDD) or a partial count area (CCHH).

P - Part of the count area (the CCHH address of the track to be copied).

F - Full count area (CCHHRKDD count of the record that was found defective).

If this parameter is omitted, P is assumed.

WRITS

track overflow segment identification.

If your data set is in the track overflow format, this identification determines recovery from errors in last records on tracks.

YES - If this is the last record on the track, it is a segment other than the last of a track overflow record.

NO - If this is the last record on the track, it is the last or only segment of a track overflow record.

If this parameter is omitted, it is assumed that it cannot be established whether a last record is a segment of an overflow record.

Using ATLAS

If a channel program encounters a unit check condition (shown in the CSW) in its execution, the I/O supervisor program will place the Sense bytes in the IOB. ATLAS can be used to recover from sense conditions shown by the following bit settings:

IOBSENS0	X'08'	Data check (except in the count area)
IOBSENS1	X'80'	Data check in the count area
IOBSENS1	X'02'	Missing address marker (see the following for combinations of this bit setting which ATLAS cannot handle.)

However, defects in the home address record or the record zero record cannot be recovered from through the use of ATLAS. These conditions are shown by:

IOBSENS1 X'02' and IOBSENS0 X'01' - home address defect.

IOBSENS1 X'0A' - record zero defect, or, home address cannot be located.

Also, before using ATLAS, you must reset error indications as follows:

NI DCBIFLGS,X'3F' Reset the DCBIFLGS error indications.

The ATLAS program will attempt to find a good alternate track and will attempt to copy the defective track onto the good track, including all error conditions in either key or data areas. The error conditions may be rectified by re-executing the channel program or through the use of the IEHATLAS utility program in a subsequent step.

Example: the following illustrates the use of the ATLAS macro instruction.

	EXCP	MYIOB	
	WAIT	ECB=MYECB	
	TM	MYECB,X'20'	TEST FOR I/O ERROR
	BO	NEXT	NO, SUCCESSFUL, GO TO ANOTHER ROUTINE
*			ROUTINE
	TM	IOBCSW+3,X'02'	UNIT CHECK
	BL	OTHER	NO, DO OTHER ERROR PROCESSING
	TM	IOBSENS0,X'08'	DATA CHECK
	BO	ATLASGO	YES, VALID ERROR
	TM	IOBSENS1,X'80'	DATA CHECK IN COUNT
	BO	ATLASGO	YES, VALID ERROR
	TM	IOBSENS1,X'0A'	MISSING ADDRESS MARKER AND NO RECORD FOUND
*			YES, ATLAS CANNOT HANDLE
*			ERROR; DO OTHER ERROR
ATLASGO EQU	BO	OTHER	PROCESSING NO, MISSING
*		*	ADDRESS MARKER ONLY.
	NI	DCB1FLGS,X'3F'	RESET ERROR INDICATORS
	ATLAS	PARMADR=THERE,CHANPRG=R	

Operation of the ATLAS Program

The ATLAS program (SVC 86):

- Establishes the availability and address of the next alternate track from the format-4 DSCB of the VTOC.
- Brings all count fields from the defective track into storage to establish the description of the track.
- Initializes the alternate track. (Write home address, write record zero.)
- Brings the key and data areas of each record into storage, one at a time, and combines them with their new count area to write the complete record onto the alternate track.
- When the copying is finished, chains the alternate to the defective track and updates the VTOC.

Control is returned to your program at the next executable instruction following the ATLAS macro instruction. The success of the ATLAS macro instruction can be determined by examining the contents of register 15, which will contain one of the return codes described below. If register 15 contains 0, 36, 40, or 44, the contents of register 0 may be significant.

**Decimal
Return
Code**

Interpretation

- 0 Successful completion. Key and data areas have been copied from the defective track onto a good alternate one. The only error encountered was in the record identified by the user's CCHHRKDD value.
- If the channel program is re-executable, it has been successfully re-executed.
- 4 This device type does not have alternate tracks that can be assigned by programming.
- 8 All alternate tracks for the device have been assigned.
- 12 A request for storage (GETMAIN macro instruction) could not be satisfied.
- 16 All attempts to initialize and transfer data to an alternate track failed. The number of attempts made is equal to 10% of the assigned alternates for the device.
- 20 The type of error shown by the sense byte cannot be handled through the use of the ATLAS macro instruction. The condition is other than a data check (in the count or data areas) or a missing address marker.
- 24 The format-4 DSCB of the VTOC cannot be read, therefore alternate track information is not available to ATLAS.
- 28 The record specified by the user was the format-4 DSCB and it could not be read.
- 32 An error found in count area of last record on the track cannot be handled because last-record-on-track identification is not supplied.
- 36 An error was encountered when reading or writing the home address record or record zero. No error recovery has taken place. If register 0 contains X'01 00 00 00', the defect is in record zero.
- 40 Successful completion. Key and data areas have been copied from the defective track onto a good alternate one. However, the alternate track may have records with defective key or data areas. Register 0 identifies the first three found defective as follows:
- n R R R
- n - Number of record numbers that follow (0, 1, 2, or 3).
- R - The number of the record found defective but copied anyhow.
- If the channel program is re-executable, it has been successfully re-executed.
- 44 Error/Errors encountered and no alternate track has been assigned. The return parameter register (register 0) will contain the R of a maximum of three error records.
- Error conditions that return this code are:
1. ATLAS received an error indication for a record with a data length in the count field of zero. Recovery was not possible because a distinction cannot be made between an EOF record and an invalid data length.

2. An error occurred while reading the count field of a record and the KDD (key length-data length) was found to be defective.
 3. More than three records on the specified track contained errors in their count fields.
- 48 No errors found on the track, no alternate assigned. ATLAS will not assign an alternate unless a track has at least one defective record.
- 52 I/O error in re-executing user's channel program. A good alternate is chained to the defective track and data has been transferred. The user's control blocks will give indication of the error condition causing failure in re-execution of his channel program.
- 56 The DCB reflects a track overflow data set, but the UCB device type shows that the device does not support track overflow.
- 60 The CCHH of the user-specified count area is not within the extents of his data set.

Figures 13 and 14 summarize the return codes that reflect track error conditions by error location.

Record in Error		Area in Error			
		Count Area		Key Area	Data Area
		CCHHR	KDD		
Record r (r ≠ 0)					
Not Last on Track		0	44	40	40
Last on Track	WRITS=YES	0	44	40	40
	WRITS=NO	0	44	40	40
	Omitted*	32	44	40	40
Record Zero					
		36	36	36	36
Home Address					
		36			

* Omitted and the Data Set is in the Track Overflow Format.

Figure 13. Error Locations and Return Codes if CCHH is in the Count Area Field

EOV – End of Volume

The EOV macro instruction identifies end-of-volume and end-of-data set conditions. For an end-of-volume condition, EOV causes switching of volumes and verification or creation of standard labels. For an end-of-data set condition, EOV causes your end-of-data set routine to be entered. Before processing trailer labels on a tape input data set, you must decrement the DCBBLKCT field. You issue EOV if switching of

magnetic tape or direct-access volumes is necessary, or if secondary allocation is to be performed for a direct-access data set opened for output.

Record in Error		Area in Error			
		Count Area		Key Area	Data Area
		CCHHR	KDD		
Record n (n=R in CCHHRKDD)					
Not Last on Track		0	0	0	0
Last on Track	WRITS=YES	0	0	0	0
	WRITS=NO	0	0	0	0
	Omitted *	32	32	0	0
Record m (m ≠ R in CCHHRKDD)					
Not Last on Track		0	44	40	40
Last on Track	WRITS=YES	0	44	40	40
	WRITS=NO	0	44	40	40
	Omitted *	32	44	40	40
Record Zero					
		36	36	36	36
Home Address					
		36			

* Omitted and the Data Set is in the Track Overflow Format.

Figure 14. Error Locations and Return Codes if CCHHRKDD is in the Count Area Field

For magnetic tape, you must issue EOV when either a tapemark is read or a reflective spot is written over. In these cases, bit settings in the 1-byte DCBOFLGS field of the data control block determine the action to be taken when EOV is executed. Before issuing EOV for magnetic tape, you must make sure that appropriate bits are set in DCBOFLGS. Bit positions 2,3,6, and 7 of DCBOFLGS are used only by the system; you are concerned with bit positions 0,1,4, and 5. The use of these DCBOFLGS bit positions is as follows:

Bit 0

set to 1 indicates that a write command was executed and that a tape mark is to be written.

Bit 1

indicates that a backward read was the last I/O operation.

Bit 4

indicates that data sets of unlike attributes are to be concatenated.

Bit 5

indicates that a tape mark has been read.

If bits 0 and 5 of DCBOFLGS are both off when EOV is executed, the tape is spaced past a tapemark, and standard labels, if present, are verified on both the old and new volumes. The direction of spacing depends on bit 1. If bit 1 is off, the tape is spaced forward; if bit 1 is on, the tape is backspaced.

If bit 0 is on when EOV is executed, a tapemark is written immediately following the last data record of the data set. Standard labels, if specified, are created on the old and the new volume.

When issuing EOV for sequentially organized output data sets on direct-access volumes, you can determine whether additional space has been obtained on the same or a different volume. You do this by checking the volume serial number in the unit control block (UCB) both before and after issuing EOV.

The only parameter of the EOV macro instruction is:

<i>[symbol]</i>	EOV	<i>dcb address</i>
-----------------	------------	--------------------

dcb address – A-type address, (2-12), or (1)
the address of the data control block that is opened for the data set. If this parameter is specified as (1), register 1 must contain this address.

CLOSE – Restore Data Control Block

The CLOSE macro instruction restores one or more data control blocks so that processing of their associated data sets can be terminated. You must issue CLOSE for all data control blocks that were used by your channel programs. Some of the procedures performed when CLOSE is executed are:

- Release of data extent block (DEB)
- Removal of information transferred to data control block fields when OPEN was executed
- Verification or creation of standard labels
- Volume disposition
- Release of programmer-written appendage routines

<i>[symbol]</i>	CLOSE	<i>(dcb address,[option],...)</i>
-----------------	--------------	-----------------------------------

dcb address – A-type address or (2-12)
the address of the data control block to be restored. More than one data control block may be specified.

option

the type of volume disposition intended for the data set. You may specify this parameter as LEAVE, REREAD, or DISP. The corresponding volume disposition when CLOSE is executed is as follows:

- LEAVE - Volume is positioned at logical end of data set.
- REREAD- Volume is positioned at logical beginning of data set.
- DISP - The disposition indicated on the DD statement is tested, and appropriate positioning is provided. This service is assumed if this operand is omitted and volume positioning is applicable. If there is no disposition specified in the DD statement when this operand is specified, LEAVE is assumed.

This parameter is ignored if specified for volumes other than magnetic tape or direct access.

Note: When CLOSE is issued for data sets on magnetic tape volumes, labels are processed according to bit settings in the DCBOFLGS field of the data control block. Before issuing CLOSE for magnetic tape, you must set the appropriate bits in DCBOFLGS. The DCBOFLGS bit positions that you are concerned with are listed in the EOVS macro instruction description. The list and execute forms of the CLOSE macro instruction are described in *OS/VS Data Management Macro Instructions*.

Control Block Fields

The fields of the input/output block, event control block, and data extent block are illustrated and explained here; the data control block fields have been described with the parameters of the DCB macro instruction in the section “EXCP Programming Specifications.”

Input/Output Block Fields

The input/output block (IOB) is not automatically constructed by a macro instruction; it must be defined as a series of constants and must be on a fullword boundary. For unit record and tape devices, the IOB is 32 bytes in length. For direct access, teleprocessing, and graphic devices, 8 additional bytes must be provided.

In Figure 15, the shaded areas indicate fields in which you must specify information. The other fields are used by the system and must be defined as all zeros. You may not place information into these fields, but you may examine them.

IOBFLAG1 (1 byte)

the type of channel program. You must set bit positions 0, 1, and 6. One bits in positions 0 and 1 indicate data chaining and command chaining, respectively. (If both data chaining and command chaining are specified, the system does not use error recovery routines except for the 2671, 1052, and 2150.) A one bit in position 6 indicates that the channel program is not related to any other channel program. Bit positions 2, 3, 4, 5, and 7 are used only by the system.

IOBFLAG2 (1 byte)

is used only by the system.

IOBSENS0 and IOBSENS1 (2 bytes)

are placed into the input/output block by the system when a unit check occurs.

IOBECBCC (1 byte)

the first byte of the completion code for the channel program. The system places this code in the high-order byte of the event control block when the channel program is posted complete. The completion codes and their meanings are listed under “Event Control Block Fields.”

IOBECBPT (3 bytes)

the address of the 4-byte event control block that you have provided.

IOBFLAG3 (1 byte)

is used only by the system.

IOBCSW (7 bytes)

the low-order seven bytes of the channel status word, which are placed into this field each time a channel end occurs.

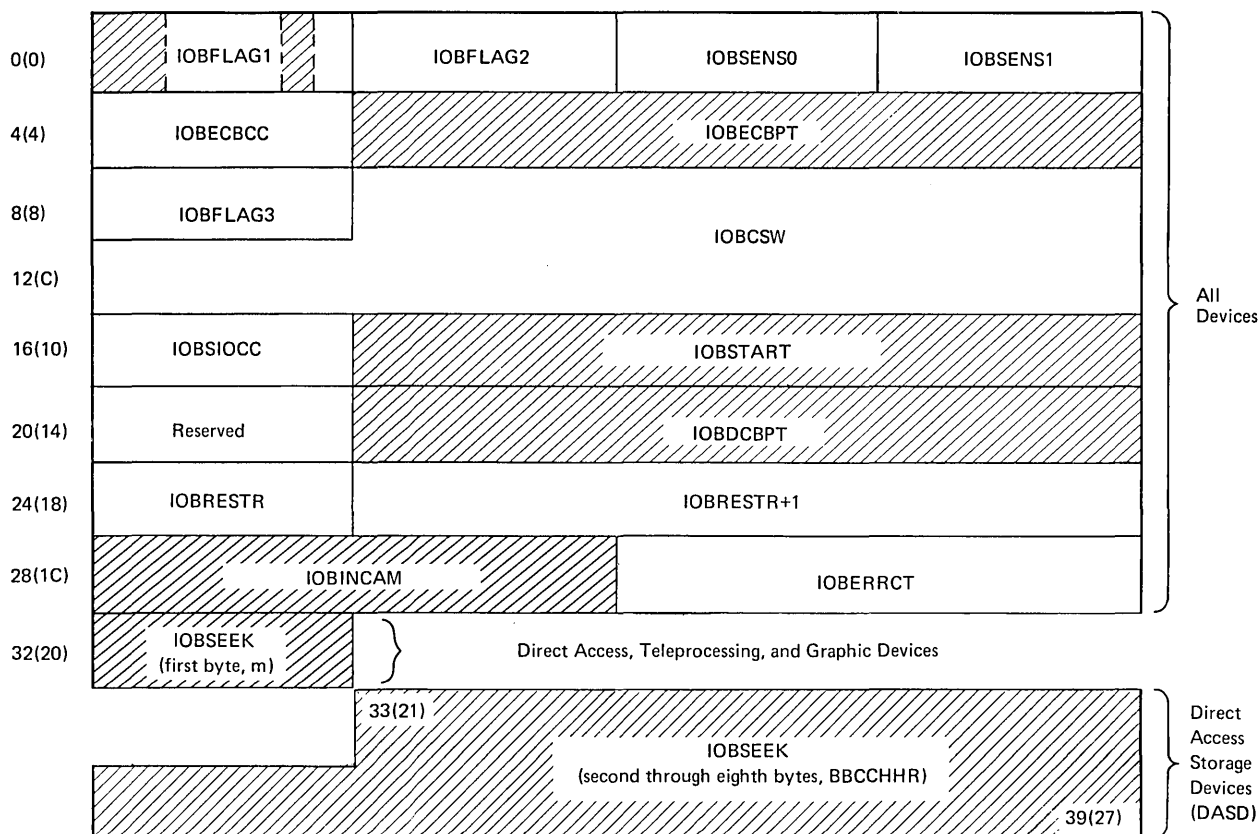


Figure 15. Input/Output Block Format

IOBSIOCC (1 byte)

in bits 0 and 1, the instruction-length code; in bits 2 and 3, the start I/O (SIO) condition code for the SIO instruction the system issues to start the channel program; and in bits 4 through 7, the program mask.

IOBSTART (3 bytes)

the starting address of the channel program to be executed.

Reserved (1 byte)

used only by the system.

IOBDCBPT (3 bytes)

the address of the data control block of the data set to be read or written by the channel program.

IOBRESTR (1 byte)

used by the system for volume repositioning in error recovery procedures.

IOBRESTR+1 (3 bytes)

used by the system to indicate the starting address of a channel program that performs special functions for error recovery procedures. The system also uses this field in procedures for making request elements available, as explained under "Error Recovery Procedures for Related Channel Programs."

IOBINCAM (2 bytes)

for magnetic tape, the amount by which the block count (DCBBLKCT) field in the device-dependent portion of the data control block is to be incremented. You may alter these bytes at any time. For forward operations, these bytes should contain a binary positive integer (usually +1); for backward operations, they should contain a binary negative integer. When these bytes are not used, all zeros must be specified.

IOBERRCT (2 bytes)

the number of retries attempted during error recovery procedures.

IOBSEEK (first byte, M)

direct-access devices: Extent entry in the data extent block that is associated with the channel program (0 indicates the first extent; 1 indicates the second, etc.).

Teleprocessing and graphic devices: The UCB index.

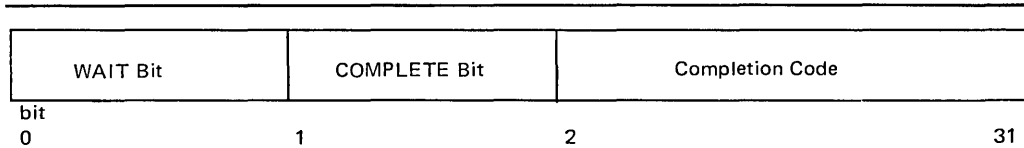
IOBSEEK (last 7 bytes, BBCCHHR)

Event Control Block Fields

You must define an event control block (ECB) as a 4-byte area on a fullword boundary. When the channel program has been completed, the input/output supervisor places a completion code containing status information into the ECB (Figure 16). Before examining this information, you must test for the setting of the "Complete Bit." If the complete bit is not on, and your problem program cannot perform other useful operations, you should issue a WAIT macro instruction that specifies the event control block. Under no circumstances may you construct a program loop that tests for the complete bit.

Data Extent Block Fields

The data extent block (DEB) is constructed by the system when an OPEN macro instruction is issued for the data control block. You may not modify the fields of the DEB, but you may examine them. For VS1, the DEB format and field description are contained in *OS/VS1 System Data Areas*. For VS2, the DEB appears in *OS/VS2 System Data Areas*.



WAIT bit

A one bit in this position indicates that the WAIT macro instruction has been issued, but that the channel program has not been completed.

COMPLETE bit

A one bit in this position indicates that the channel program has been completed; If it has not been completed, a zero bit is in this position.

Completion Code

This code, which includes the wait and complete bits, may be one of the following 4-byte hexadecimal expressions:

Code	Interpretation
7F000000	Channel program has terminated without error.
41000000	Channel program has terminated with permanent error.
42000000	Channel program has terminated because a direct-access extent address has been violated.
44000000	Channel program has been intercepted because of permanent error associated with device end for previous request. You may reissue the intercepted request.
48000000	Request element for channel program has been made available after it has been purged.
4B000000	One of the following errors occurred during tape error recovery processing. <ul style="list-style-type: none"> • The CSW command address in the IOB is zeros. • An unexpected load point was encountered.
4F000000	Error recovery routines have been entered because of direct-access error but are unable to read home address or record 0.

Figure 16. Event Control Block After Posting of Completion Code (EXCP)

USING XDAP TO READ AND WRITE TO DIRECT-ACCESS DEVICES

The execute direct-access program (XDAP) macro instruction provides you with a means of reading, verifying, or updating blocks on direct-access volumes without using an access method and without writing your own channel program. This chapter explains what the XDAP macro instruction does and how you can use it. The control block generated when XDAP is issued and the macro instruction used with XDAP are also discussed.

Since most of the specifications for XDAP are similar to those for the execute channel program (EXCP) macro instruction, you should be familiar with the “Executing Your Own Channel Programs (EXCP)” chapter of this publication, as well as with the information contained in *OS/VS Data Management Services Guide, GC26-3783*, which provides how-to information for using the access method routines of the system control program.

Introduction

Execute direct-access program (XDAP) is a macro instruction that you may use to read, verify, or update a block on a direct-access volume. If you are not using the standard IBM data access methods, you can, by issuing XDAP, generate the control information and channel program necessary for reading or updating the records of a data set.

You cannot use XDAP to add blocks to a data set, but you can use it to change the keys of existing blocks. Any block configuration and any data set organization can be read or updated.

Although the use of XDAP requires much less storage than do the standard access methods, it does not provide many of the control program services that are included in the access methods. For example, when XDAP is issued, the system does not block or deblock records and does not verify block length.

To issue XDAP, you must provide the actual device address of the track containing the block to be processed. You must also provide either the block identification or the key of the block, and specify which of these is to be used to locate the block. If a block is located by identification, both the key and data portions of the block may be read or updated. If a block is located by key, only the data portion can be processed.

For additional control over I/O operations, you may write appendages, which must be entered into the SVC library. Descriptions of these routines and their coding specifications are contained in the “Executing Your Own Channel Programs (EXCP)” section of this publication.

XDAP Requirements

Before issuing the XDAP macro instruction, you must issue a DCB macro instruction, which produces a data control block (DCB) for the data set to be read or updated. You must also issue an OPEN macro instruction, which initializes the data control block and produces a data extent block (DEB).

When the XDAP macro instruction is issued, another control block, containing both control information and executable code, is generated. This control block may be logically divided into three sections:

- An event control block (ECB), which is supplied with a completion code each time the direct-access channel program is terminated.
- An input/output block (IOB), which contains information about the direct-access channel program.
- A direct-access channel program, which consists of three or four channel command words (CCWs). The type of channel program generated depends on specifications in the parameters of the XDAP macro instruction.

After this XDAP control block is constructed, the direct-access channel program is executed. A block is located by either its actual address or its key, and is either read or updated.

When the channel program has terminated, a completion code is placed into the event control block. After issuing XDAP, you should therefore issue a WAIT macro instruction specifying the event control block to determine whether the direct-access program has terminated. If volume switching is necessary, you must issue an EOVS macro instruction. When processing of the data set has been completed, you must issue a CLOSE macro instruction to restore the data control block.

Macro Specifications for Use With XDAP

When you are using the XDAP macro instruction, you must also issue DCB, OPEN, CLOSE, and, in some cases, the EOVS macro instructions. The parameters of the XDAP macro instruction are listed and described here. For the other required macro instructions, special requirements or options are explained, but you should refer to “Macro Specifications for Use with EXCP” for listings of their parameters.

DCB – Define Data Control Block

The EXCP form of the DCB macro instruction produces a data control block that can be used with the XDAP macro instruction. You must issue a DCB macro instruction for each data set to be read or updated by the direct-access channel program. The section “DCB – Define Data Control Block for EXCP” contains a diagram of the data control block, as well as a listing of the parameters of the DCB macro instruction.

OPEN – Initialize Data Control Block

The OPEN macro instruction initializes one or more data control blocks so that their associated data sets can be processed. You must issue OPEN for all data control blocks that are to be used by the direct-access program. Some of the procedures performed when OPEN is executed are:

- Construction of data extent block (DEB).
- Transfer of information from DD statements and data set labels to the data control block.
- Verification or creation of standard labels.
- Loading of programmer-written appendage routines.

The two parameters of the OPEN macro instruction are the address(es) of the data control block(s) to be initialized, and the intended method of I/O processing of the data set. The method of processing may be specified as either INPUT or OUTPUT; however, if neither is specified, INPUT is assumed.

XDAP – Execute Direct-Access Program

The XDAP macro instruction produces the XDAP control block (i.e., the ECB, IOB, and channel program) and executes the direct-access channel program. The format of the XDAP macro instruction is:

<i>[symbol]</i>	XDAP	<i>ecb-symbol,type,dcb-addr,area-addr,length-value, (key-addr,keylength-value)],blkref-addr,[sector-addr], [MF = E L]</i>
-----------------	-------------	---------------------------------------------------------------------------------------------------------------------------------------

ecb-symbol – symbol or (2-12)

the symbolic name to be assigned to the XDAP control block. Registers can be used only with MF=E or MF=L.

type – {RI}
 {RK}
 {WI}
 {WK}
 {VI}
 {VK}

the type of I/O operation intended for the data set and the method by which blocks of the data set are to be located. Two values must be coded in this field. The following combinations are valid: RI, RK, WI, WK, VI, and VK.

The codes and their meanings are:

- R - Read a block.
- W - Write a block.
- V - Verify contents of a block but do not transfer data.
- I - Locate a block by identification. (The key portion, if present, and the data portion of the block are read or written.)
- K - Locate a block by key. (Only the data portion of the block is read or written.) If you code this value, you must code the *key-addr* and *key-length-value* operands.

dcb-addr – A-type address or (2-12)

the address of the data control block of the data set.

area-addr – A-type address or (2-12)

the address of an input or output area for a block of the data set.

length-value – absexp or (2-12)

the number of bytes to be transferred to or from the input or output area. If blocks are to be located by identification and the data set contains keys, the value must include the length of the key. The maximum number of bytes transferred is 32767.

key-addr – RX-type address or (2-12)

when blocks are to be located by key, the address of a virtual storage field that contains the key of the block to be read or overwritten.

keylength-value – absexp or (2-12)

when blocks are to be located by key, the length of the key. The maximum length is 255 bytes.

blkref-addr – RX-type address or (2-12)

the address of a field in virtual storage containing the actual device address of the track containing the block to be located. The actual address of a block is in the form **MBBCCCHR**, where **M** indicates which extent entry in the data extent block is associated with the direct-access program; **BB** is not used but must be zero; **CC** indicates the cylinder address; **HH** indicates the actual track address; and **R** indicates the block identification. **R** is not used when blocks are to be located by key. (See “Conversion of Relative Block Address to Actual Device Address” later in this chapter for more detailed information.)

sector-addr – RX-type address or (2-12)

the address of a 1-byte field containing a sector value. The sector-address parameter is used for rotational position sensing (RPS) devices only. The parameter is optional, but its use will improve channel performance. When the parameter is coded, a set-sector CCW (using the sector value indicated by the data address field) precedes the Search-ID-Equal command in the channel program. The sector-address parameter is ignored if the type parameter is coded as RK, WK, or VK, or is omitted in the execute form of the XDAP macro instruction.

Note: No validity check is made on either the address or the sector value when the XDAP macro is issued. However, a unit exception interrupt will occur during the channel program execution if the sector value is larger than the maximum for the device or if the sector-addr operand is used when accessing a device without RPS. (See “Obtaining Sector Number of a Block or a Device with the RPS Feature” later in this chapter for more detailed information.)

MF=

you may use the L-form of the XDAP macro instruction for a macro expansion consisting of only a parameter list, or the E-form for a macro expansion consisting of only executable instructions.

MF=L

The first two operands (*ecb-symbol* and *type*) are required and must be coded as symbols. All other operands are optional except *blkref-addr*, which is ignored if coded. The last five operands must be coded as A-type addresses.

MF=E

The first operand (*ecb-symbol*) is required and may be coded as a symbol or supplied in register 2-12. The *type*, *dcb-addr*, *area-addr*, and *length-value* operands may be supplied in either the L- or E-form. The *blkref-addr* operand may be supplied in the E-form or moved into the IOB by you. The *sector-addr* is optional; it may be coded either in both the L- and E-form or in neither.

The *dcb-addr*, *area-addr*, *blkref-addr*, and *sector-value* operands may be coded as RX-type addresses or supplied in register 2-12. The *length-value* and *keylength-value* operands can be specified as a decimal digit or supplied in register 2-12.

EOV – End of Volume

The EOV macro instruction identifies end-of-volume and end-of-data set conditions. For an end-of-volume condition, EOV causes switching of volumes and verification or creation of standard labels. For an end-of-data set condition, EOV causes your end-of-data set routine to be entered. When using XDAP, you issue EOV if switching of direct-access volumes is necessary, or if secondary allocation is to be performed for a direct-access data set opened for output.

The only parameter of the EOV macro instruction is the address of the data control block of the data set.

CLOSE – Restore Data Control Block

The CLOSE macro instruction restores one or more data control blocks so that processing of their associated data sets can be terminated. You must issue CLOSE for all data sets that were used by the direct-access channel program. Some of the procedures performed when CLOSE is executed are:

- Release of data extent block (DEB)
- Removal of information transferred to data control block fields when OPEN was executed
- Verification or creation of standard labels
- Release of programmer-written appendage routines

The only parameter of the CLOSE macro instruction is the address of the data control block to be restored. (More than one data control block may be specified.)

Control Blocks Used with XDAP

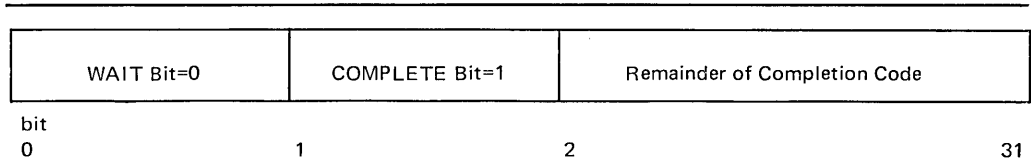
The three control blocks generated during execution of the XDAP macro instruction are described here.

Event Control Block

The event control block (ECB) begins on a fullword boundary and occupies the first 4 bytes of the XDAP control block. Each time the direct-access channel program terminates, the input/output supervisor places a completion code containing status information into the event control block (Figure 17). Before examining this information, you must test for the setting of the “COMPLETE Bit” by issuing a WAIT macro instruction specifying the event control block.

Input/Output Block

The input/output block (IOB) is 40 bytes in length and immediately follows the event control block. The “Control Block Guide” section in the EXCP section of this publication contains a diagram of the input/output block. The only fields with which the user of XDAP is concerned are the IOBSENS0, IOBSENS1, and IOBCSW fields (see Figure 13). You may wish to examine these fields when a unit check condition or an I/O interruption occurs.



WAIT Bit

A one bit in this position indicates that the direct-access channel program has not been completed.

COMPLETE Bit

A one bit in this position indicates that the channel program has been completed; if it has not been completed, a zero bit is in this position.

Completion Code

This code, which includes the WAIT and COMPLETE bits, may be one of the following 4-byte hexadecimal expressions:

Code	Interpretation
7F000000	Direct-access program has terminated without error.
41000000	Direct-access program has terminated with permanent error.
42000000	Direct-access program has terminated because a direct-access extent address has been violated.
44000000	Channel program has been intercepted because of permanent error associated with device end for previous request. You may reissue the intercepted request.
48000000	Request element for channel program has been made available after it has been purged.
4F000000	Error recovery routines have been entered because of direct-access error but are unable to read home address or record 0.

Figure 17. Event Control Block After Posting of Completion Code (XDAP)

Direct-Access Channel Program

The direct-access channel program is 24 bytes in length (except when set sector is used for RPS devices) and immediately follows the input/output block. Depending on the type of I/O operation specified in the XDAP macro instruction, one of four channel programs may be generated. The three channel command words for each of the four possible channel programs are shown in Figure 18.

When a sector address is specified with an RI, VI, or WI operation, the channel program is 32 bytes in length. Each of the channel programs in Figure 18 would be, in this case, preceded by a set sector command.

Type of I/O Operation	CCW	Command Code
Read by identification	1	Search ID Equal
	2	Transfer in Channel
	3	Read Key and Data
Verify by identification ¹	1	Search Key Equal
	2	Transfer in Channel
	3	Read Data
Write by identification	1	Search ID Equal
	2	Transfer in Channel
	3	Write Key and Data
Write by key	1	Search Key Equal
	2	Transfer in Channel
	3	Write Data

¹For verifying operations, the third CCW is flagged to suppress the transfer of information to virtual storage.

Figure 18. The XDAP Channel Programs

Conversion of Relative Block Address to Actual Device Address

To issue XDAP, you must provide the actual device address of the track containing the block to be processed. If you know only the relative block address, you can convert it to the actual address by using a resident system routine. The entry point to this conversion routine is labeled IECPCNVT. The address of the entry point (CVTPCNVT) is in the communication vector table (CVT). The address of the CVT is in location 16. (For the displacements and descriptions of the CVT fields, see *OS/VS1 System Data Areas* for VS1 systems and *OS/VS2 System Data Areas* for VS2 systems.)

The conversion routine does all its work in general registers. You must load registers 0, 1, 2, 14, and 15 with input to the routine. Register usage is as follows:

Register	Use
0	Must be loaded with a 4-byte value of the form TTRN, where TT is the number of the track relative to the beginning of the data set, R is the identification of the block on that track, and N is the concatenation number of the data set. (0 indicates the first or only data set in the concatenation, 1 indicates the second, etc.)
1	Must be loaded with the address of the data extent block (DEB) of the data set.
2	Must be loaded with the address of an 8-byte area that is to receive the actual address of the block to be processed. The converted address is of the form MBBCCHHR, where M indicates which extent entry in the data extent block is associated with the direct-access program (0 indicates the first extent, 1 indicates the second, etc.); BB indicates the bin number of the direct-access volume; CC indicates the cylinder address; HH indicates the actual track address; and R indicates the block identification.
3-8	Are not used by the conversion routine.
9-13	Are used by the conversion routine and are not restored.

- 14 Must be loaded with the address to which control is to be returned after execution of the conversion routine.
- 15 Is used by the conversion routine as a base register and must be loaded with the address at which the conversion routine is to receive control.

Obtaining Sector Number of a Block on a Device With the RPS Feature

To obtain the performance improvement given by rotational position sensing, you should specify the *sector-addr* parameter on the XDAP macro. For programs which may be used for both RPS and non-RPS devices, the UCBTYP field can be checked to determine whether or not the device has the rotational position sensing feature.

The *sector-addr* parameter on the XDAP macro specifies the address of a one byte field in your region. You must store the sector number of the block to be located in this field. You can obtain the sector number of the block by using a resident conversion routine, IEC0SCR1. The address of this routine is in field CVT0SCR1 of the CVT, and the address of th CVT is in location 16. The routine should be invoked via a BALR 14, 15 instruction.

For RPS devices, the conversion routine does all its work in general registers. You must load registers 0, 2, 14, and 15 with input to the routine. Register usage is as follows:

Register	Use
0	<p>For fixed-length records, register 0 must be loaded with a 4-byte value in the form DDKR, where DD is a 2-byte field containing the physical block size, K is a 1-byte field containing the key length, and R is a 1-byte field containing the number of the record for which a sector value is desired. The high-order bit of register 0 must be turned off (set to 0) to indicate fixed-length records.</p> <p>For variable-length records, register 0 must be loaded with a 4-byte value in the form BBIR, where BB is the total number of key and data bytes up to, but not including, the target record; I is a 1-byte key indicator (1 for keyed records, 0 for records without keys); and R is a 1-byte field containing the number of the record for which a sector value is desired. The high-order bit of register 0 must be turned on (set to 1) to indicate variable-length records.</p>
1	Not used by the sector convert routine.
2	Must be loaded with a 4-byte field in which the first byte is the UCB device type code for the device (obtainable from UCB+19), and the remaining three bytes are the address of a 1-byte area that is to receive the sector value.
3-8,12,13	Not used.
9-11	Used by the convert routine and are not saved or restored.
14	Must be loaded with the address to which control is to be returned after execution of the sector conversion routine.
15	Used by the conversion routine as a base register and must be loaded with the address of the entry point to the conversion routine.

PASSWORD PROTECTING YOUR DATA SETS

To use the data set protection feature of the operating system, you must create and maintain a PASSWORD data set consisting of records that associate the names of the protected data sets with the password assigned to each data set. There are four ways to maintain the PASSWORD data set:

- You can write your own routines.
- You can use the PROTECT macro instruction.
- You can use the utility control statements of the IEHPROGM utility program.
- For OS/VS2 systems with TSO, you can use the TSO PROTECT command.

This chapter discusses only the first two of the four ways – it provides technical detail about the PASSWORD data set that is necessary for writing your own routines, and it describes how to use the PROTECT macro instruction. (The last two of the four ways are discussed in other publications, as indicated in the list of publications below.)

Before using the information in this chapter, you should be familiar with information in several related publications. The following publications are recommended:

- *OS/VS Data Management Services Guide*, GC26-3783, contains a general description of the data set protection feature.
- *OS/VS Message Library: VS1 System Messages*, GC38-1001, contains a description of the operator messages and replies associated with the data set protection feature for VS1.
- *OS/VS Message Library: VS2 System Messages*, GC28-1002, contains a description of the operator messages and replies associated with the data set protection feature for VS2.
- *OS/VS JCL Reference*, GC28-0618, contains a description of the data definition (DD) statement parameter used to indicate that a data set is to be password protected.
- *OS/VS DADSM Logic*, SY26-3787, contains a description of the PASSWORD data set record format.
- *OS/VS Utilities*, GC35-0005, contains a description of how to maintain the PASSWORD data set using the utility control statements of the IEHPROGM utility program.
- *OS/VS2 TSO Command Language Reference*, GC28-0646, describes the use of the TSO PROTECT command.

Introduction

In addition to the usual label protection that prevents opening of a data set without the correct data set name, the operating system provides data set security options that prevent unauthorized access to confidential data. Password protection prevents access to data sets, until a correct password is entered by the system operator, or, for TSO, a remote terminal operator.

The following are the types of access allowed to password protected data sets:

- PWREAD/PWRITE - A password is required to read or write.
- PWREAD/NOWRITE - A password is required to read. Writing is not allowed.
- NOPWREAD/PWRITE - Reading is allowed without a password. A password is required to write.

To prepare for use of the data set protection feature of the operating system, you place a sequential data set, named **PASSWORD**, on the system residence volume. This data set must contain at least one record for each data set placed under protection. In turn, each record contains a data set name, a password for that data set, a counter field, a protection mode indicator, and a field for recording any information you desire to log. On the system residence volume, these records are formatted as a “key area” (data set name and password) and a “data area” (counter field, protection mode indicator, and logging field). The data set is searched on the “key area.”

You can write routines to create and maintain the **PASSWORD** data set. If you use the **PROTECT** macro instruction to maintain the **PASSWORD** data set, see the section in this chapter called “Using the Macro Instruction to Maintain the **PASSWORD** Data Set.” If you use the **IEHPROGM** utility program to maintain the **PASSWORD** data set, see *OS/VS Utilities*. These routines may be placed in your own library or the system’s linkage editor library (**SYS1.LINKLIB**). You may use a data management access method or **EXCP** programming to read from and write to the **PASSWORD** data set.

If a data set is to be placed under protection, it must have a protection indicator set in its label (format-1 **DSCB** or header 1 tape label). This is done by the operating system when the data set is created, by the **IEHPROGM** utility program, or, by the **PROTECT** macro when creating or adding the control password. The protection indicator is set in response to a value in the **LABEL=** operand of the **DD** statement associated with the data set being placed under protection. *OS/VS JCL Reference* describes the **LABEL** operand.

Note: Data sets on magnetic tape are protected only when standard labels are used.

Password-protected data sets can only be accessed by programs that can supply the correct password. When the system control program receives a request to open a protected data set, it issues a message that requests that the password be given. The message goes to the operator console, or, if the program requesting that the data set be opened is running under **TSO** in the foreground, to the **TSO** terminal operator. If you want the password supplied by another method in your installation, you can modify the **READPSWD** source module or code a new routine to replace **READPSWD** in **SYS1.SVCLIB**.

PASSWORD Data Set Characteristics

The **PASSWORD** data set must reside on the same volume as your operating system. The space you allocate to the **PASSWORD** data set must be contiguous, i.e., its **DSCB** must indicate only one extent. The amount of space you allocate depends on the number of data sets your installation wants to protect. Each entry in the **PASSWORD** data set requires 132 bytes of space. The organization of the **PASSWORD** data set is physical sequential, the record format is unblocked, fixed-length records (**RECFM=F**). These records are 80 bytes long (**LRECL=80, BLKSIZE=80**) and form the data area of the **PASSWORD** data set records on direct-access storage. In these direct-access storage records, the data area is preceded by a key area of 52 bytes (**KEYLEN=52**). The key area contains the fully qualified data set name of up to 44 bytes and a

password of one to eight bytes, left justified with blanks added to fill the areas. The password assigned may be from one to eight alphanumeric characters in length. *OS/VS DADSM Logic*, describes the PASSWORD data set record format.

You can protect the PASSWORD data set itself by creating a password record for it when your program initially builds the data set. Thereafter, the PASSWORD data set cannot be opened (except by the operating system routines that scan the data set) unless the operator enters the password.

Note: If a problem occurs on a password-protected system data set, maintenance personnel must be provided with the password in order to access the data set and resolve the problem.

Creating Protected Data Sets

A data definition (DD) statement parameter (LABEL=) is used to indicate that a data set is to be placed under protection. Operating procedures at your installation must ensure that password records for all data sets currently under protection are entered in the PASSWORD data set. You may, for example, create a data set and set the protection indicator in its label, without entering a password record for it in the PASSWORD data set. However, once the data set is closed, any subsequent attempt to open results in termination of the program attempting to open the data set, unless the password record is available and the operator can honor the request for the password. (Note that if the protection module is NOPWREAD and the request is to open the data set for input, no password will be required.)

Protection Feature Operating Characteristics

The topics that follow provide information concerning actions of the protection feature in relation to termination of processing, volume switching, data set concatenation, SCRATCH and RENAME functions, and counter maintenance.

Termination of Processing

Processing is terminated when:

1. The operator cannot supply the correct password for the protected data set being opened after two tries.
2. A password record does not exist in the PASSWORD data set for the protected data set being opened.
3. The protection mode indicator in the password record, and the method of I/O processing specified in the Open routine do not agree, e.g., OUTPUT specified against a read-only protection mode indicator.
4. There is a mismatch in data set names for a data set involved in a volume switching operation. This is discussed in the next topic.

Volume Switching

The operating system end-of-volume routine does not request a password for a data set involved in a volume switch. Continuity of protection is handled in the following ways:

Input Data Sets – Tape and Direct-Access Devices: Processing continues if the data set name in the tape label or DSCB on the volume switched to matches the name of the data set opened with the password. If they do not match, processing is terminated.

Output Data Sets – Tape Devices: The protection indicator in the tape label for the first data set on the volume switched to is tested:

1. If the protection indicator is set ON and the data set name in the label and the name of the data set opened with the password match, processing continues. An unequal comparison results in a call for another volume.
2. If the protection indicator is OFF, processing continues, and a new label is written with the protection indicator set ON.
3. If only a volume label exists on the volume switched to, processing continues, and a new label is written with the protection indicator set ON.

Output Data Sets – Direct-Access Devices: For existing data sets, if the data set name in a DSCB on the volume switched to and the name of the data set opened with the password match, processing continues. For new output data sets, the volume switching mechanism ensures continuity of protection: the DSCB created on the volume switched to will indicate protection.

Data Set Concatenation

A password is requested for *every* protected data set that is involved in a concatenation of data sets, regardless of whether the other data sets involved are protected or not.

SCRATCH and RENAME Functions

Each attempt to delete or rename a protected data set results in a request for the password via the IEC301A message. The password supplied in response to this message must be associated with a “WRITE” protection mode indicator.

Counter Maintenance

The operating system does not maintain the counter in the password record and no overflow indication will be given (overflow after 65,535 openings). You must provide a counter maintenance routine to check and, if necessary, reset this counter.

Using the PROTECT Macro Instruction to Maintain the PASSWORD Data Set

To use the PROTECT macro instruction, your PASSWORD data set must be on the system residence volume. The PROTECT macro can be used to:

- Add an entry to the PASSWORD data set.
- Replace an entry in the PASSWORD data set.
- Delete an entry from the PASSWORD data set.
- Provide a list of information about an entry in the PASSWORD data set; this list will contain the security counter, access type, and the 77 bytes of security information in the “data area” of the entry.

In addition, the PROTECT macro, updates the DSCB of a protected direct-access data set to reflect its protection status; this feature eliminates the need for you to use job control language whenever you protect a data set.

PASSWORD Data Set Characteristics and Record Format When You Use the PROTECT Macro Instruction

When you use the PROTECT macro, the record format and characteristics of the PASSWORD data set are no different from the record format and characteristics that apply when you use your own routines to maintain it.

Number of Records for Each Protected Data Set

When you use the PROTECT macro, the PASSWORD data set must contain at least one record for each protected data set. The password (the last 8 bytes of the “key area”) that you assign when you protect the data set for the first time is called the *control password*. In addition, you may create as many secondary records for the same protected data set as you need. The passwords assigned to these additional records are called *secondary passwords*. This feature is helpful if you want several users to have access to the same protected data set, but you also want to control the manner in which they can use it. For example: one user could be assigned a password that allowed the data set to be read and written, and another user could be assigned a password that allowed the data set to be read only.

Note: The PROTECT macro will update the protection mode indicator in the format-1 DSCB in the protected data set only when you issue it for adding, replacing, or deleting a control password.

Protection Mode Indicator

You can set the protection mode indicator in the password record to four different values:

- X'00' to indicate that the password is a secondary password and the protected data set is to be read only (PWREAD).
- X'80' to indicate that the password is the control password and the protected data set is to be read only (PWREAD).
- X'01' to indicate that the password is a secondary password and the protected data set is to be read and written (PWREAD/PWRITE).
- X'81' to indicate that the password is the control password and the protected data set is to be read and written (PWREAD/PWRITE).

Because the DSCB of the protected data set is updated only when the control password is changed, you may request protection attributes for secondary passwords that conflict with the protection attributes of the control password.

Because of the sequence in which the protection status of a data set is checked, the following defaults will occur:

If control password is:	Secondary password must be:
1. PWREAD/PWRITE or PWREAD/NOWRITE	PWREAD/PWRITE or PWREAD/NOWRITE
2. NOPWREAD/PWRITE	NOPWREAD/PWRITE

If the control password is set to either of the settings in item 1 above, the secondary password will be set to PWREAD/PWRITE if you try to set it to NOPWREAD/PWRITE.

If the control password is changed from either of the settings in item 1 to the setting in item 2 above, the secondary password will be automatically reset to NOPWREAD/PWWRITE.

If the control password is changed from the setting in item 2 to either of the settings in item 1 above, the secondary password is set by the system to PWREAD/PWWRITE.

PROTECT Macro Specification

The format is:

[symbol] **PROTECT** *parameter list address*

parameter list address – A-type address, (2-12), or (1)

indicates the location of the parameter list. The parameter list must be set up before the PROTECT macro is issued. The address of the parameter list may be passed in register 1, in registers 2 through 12, or as an A-type address. The first byte of the parameter list must be used to identify the function (add, replace, delete, or list) you want to perform. See Figures 19 through 22 for the parameter lists and codes used to identify the functions.

0	X'01'	1	00 00 00
4	Data Set Length	5	Pointer to Data Set Name
8	00	9	00 00 00
12	00	13	Pointer to Control Password
16	Number of Volumes	17	Pointer to Volume List
20	Protection Code	21	Pointer to New Password
24	String Length	25	Pointer to String

- 0 X'01'
Entry code indicating ADD function.
- 13 Pointer to control password.
The control password is the password assigned when the data set was placed under protection for the first time. The pointer can be 3 bytes of binary zeros if the new password is the control password.
- 16 Number of volumes.
If the data set is not cataloged and you want to have it flagged as protected, you have to specify the number of volumes in this field. A zero indicates that the catalog information should be used.
- 17 Pointer to volume list.
If the data set is not cataloged and you want to have it flagged as protected, you provide the address of a list of volume serial numbers in this field. Zeros indicate that the catalog information should be used.
- 20 Protection code.
A one-byte number indicating the type of protection: X'00' indicates default protection (for the ADD function; the default protection is the type of protection specified in the control password record of the data set); X'01' indicates that the data set is to be read and written; X'02' indicates that the data set is to be read only; and X'03' indicates that the data set can be read without a password, but a password is needed to write into it. The PROTECT macro will use the protection code value, specified in the parameter list, to set the protection mode indicator in the password record.
- 21 Pointer to new password.
If the data set is being placed under protection for the first time, the new password becomes the control password. If you are adding a secondary entry, the new password is different from the control password.
- 24 String length.
The length of the character string (maximum 77 bytes) that you want to place in the optional information field of the password record. If you don't want to add information, set this field to zero.
- 25 Pointer to string.
The address of the character string that is going to be put in the optional information field. If you don't want to add additional information, set this field to zero.

Figure 19. Parameter List for ADD Function

0	X'02'	1	00 00 00
4	Data Set Length	5	Pointer to Data Set Name
8	00	9	Pointer to Current Password
12	00	13	Pointer to Control Password
16	Number of Volumes	17	Pointer to Volume List
20	Protection Code	21	Pointer to New Password
24	String Length	25	Pointer to String

- 0 X'02'
Entry code indicating REPLACE function.
- 9 Pointer to current password.
The address of the password that is going to be replaced.
- 13 Pointer to control password.
The address of the password assigned to the data set when it was first placed under protection. The pointer can be set to 3 bytes of binary zero if the current password is the control password.
- 16 Number of volumes.
If the data set is not cataloged and you want to have it flagged as protected, you have to specify the number of volumes in this field. A zero indicates that the catalog information should be used.
- 17 Pointer to volume list.
If the data set is not cataloged and you want to have it flagged as protected, you have to provide the address of a list of volume serial numbers in this field. If this field is zero, the catalog information will be used.
- 20 Protection code.
A one-byte number indicating the type of protection: X'00' indicates that the protection is default protection (for the REPLACE function the default protection is the protection specified in the current password record of the data set); X'01' indicates that the data set is to be read and written; X'02' indicates that the data set is to be read only; and X'03' indicates that the data set can be read without a password, but a password is needed to write into the data set.
- 21 Pointer to new password.
The address of the password that you want to replace the current password.
- 24 String length.
The length of the character string (maximum 77 bytes) that you want to place in the optional information field of the password record. Set this field to zero if you don't want to add additional information.
- 25 Pointer to string.
The address of the character string that is going to be put in the optional information field of the password record. Set the address to zero if you don't want to add additional information.

Figure 20. Parameter List for REPLACE Function

0	X'03'	1	00 00 00
4	Data Set Length	5	Pointer to Data Set Name
8	00	9	Pointer to Current Password
12	00	13	Pointer to Control Password
16	Number of Volumes	17	Pointer to Volume List

- 0 X'03'.
Entry code indicating DELETE function.
- 9 Pointer to current password.
The address of the password that you want to delete. You can delete either a control entry or a secondary entry.
- 13 Pointer to control password.
The address of the password assigned to the data set when it was placed under protection for the first time. The pointer can be 2 bytes of binary zero if the current password is also the control password.
- 16 Number of volumes.
If the data set is not cataloged and you want to have it flagged as protected, you have to specify the number of volumes in this field. A zero indicates that the catalog information should be used.
- 17 Pointer to volume list.
If the data set is not cataloged and you want to have it flagged as protected, you have to provide the address of a list of volume serial numbers in this field. If this field is zero, the catalog information will be used.

Figure 21. Parameter List for DELETE Function

0	X'04'	1	Pointer to 80 Byte Buffer
4	Data Set Length	5	Pointer to Data Set Name
8	00	9	Pointer to Current Password

- 0 X'04'.
Entry code indicating LIST function.
- 1 Address of 80-byte buffer.
The address of a buffer where the list of information can be returned to your program by the macro instruction.
- 9 Pointer to current password.
The address of the password of the record that you want listed.

Figure 22. Parameter List for LIST Function

Return Codes From the PROTECT Macro

When the PROTECT macro finishes processing, register 15 contains a return code that indicates what happened during the processing. Figure 23 contains the return codes and their interpretation.

Register 15	Interpretation
0	The updating of the PASSWORD data set was successfully completed.
4	The PASSWORD of the data set name was already in the password data set.
8	The password of the data set name was not in the PASSWORD data set.
12	A control password is required or the one supplied is incorrect.
16	The supplied parameter list was incomplete or incorrect.
20	There was an I/O error in the PASSWORD data set.
**24	The PASSWORD data set was full.
28	The validity check of the buffer address failed.
*32	The LOCATE macro failed. LOCATE's return code is in register 1, and the number of indexes searched is in register 0.
*36	The OBTAIN macro failed. OBTAIN's return code is in register 1.
*40	The DSCB could not be updated.
44	The PASSWORD data set does not exist.
*48	Tape data set cannot be protected.
*52	Data set in use.

*For these return codes, the PASSWORD data set has been updated, but the DSCB has not been flagged to indicate the protected status of the data set.

**For this return code, a message is written to the console indicating that the PASSWORD data set is full.

Figure 23. Return Codes from the PROTECT Macro Instruction

SYSTEM MACRO INSTRUCTIONS

This chapter describes miscellaneous macro instructions that allow you either to modify control blocks or to obtain information from control blocks and system tables.

Before reading this chapter, you should be familiar with the information in the following publications:

- *OS/VS and DOS/VS Assembler Language*, GC33-4010, contains the information necessary to code programs in the assembler language.
- *OS/VS1 System Data Areas*, SY28-0605, contains format and field descriptions of the VS1 system control blocks referred to in this chapter.
- *OS/VS2 System Data Areas*, SY26-0606, contains format and field descriptions of the VS2 system control blocks referred to in this chapter.

Introduction

The system macro instructions are described in these functional groupings:

- Mapping (IEFUCBOB, IEFJFCBN, and CVT)
- Obtaining device characteristics (DEVTYPE)
- Manipulating the JFCB (RDJFCB)
- Data security (DEBCHK)
- Manipulating queues (PURGE and RESTORE)

Mapping System Data Areas

The IEFUCBOB, IEFJFCBN, and CVT macro instructions are used as DSECT expansions that define the symbolic names of fields within the unit control block (UCB), job file control block (JFCB), and communication vector table (CVT), respectively. When coding these instructions, you must precede each with a DSECT statement.

The IEFUCBOB and IEFJFCBN macro definitions are in the macro library (SYS1.AMACLIB) ready for use. The CVT definition is in a system generation library (SYS1.AMODGEN) and must be copied (using IEBCOPY) into SYS1.AMACLIB, or SYS1.AMODGEN may be concatenated to the macro library before reference can be made to it.

The fields in these blocks are shown and described in *OS/VS System Data Areas* for VS1 systems and in *OS/VS2 System Data Areas* for VS2 systems.

IEFUCBOB – Mapping the UCB

This macro instruction defines the symbolic names of all fields in the unit control block (UCB). Code this macro instruction with blank name and operand fields, and precede it with a DSECT statement.

The format is:

[symbol]	DSECT
	IEFUCBOB

IEFJFCBN – Mapping the JFCB

This macro instruction defines the symbolic names of all fields in the job file control block (JFCB). Code this macro instruction with blank name and operand fields, and precede it with a DSECT statement.

The format is:

<i>[symbol]</i>	DSECT
	IEFJFCBN

CVT – Mapping the CVT

This macro instruction defines the symbolic names of all fields in the communication vector table (CVT). Code this macro instruction with blank name and operand fields, and precede it with a DSECT statement.

The format is:

<i>[symbol]</i>	DSECT
	CVT

Obtaining I/O Device Characteristics

Use the DEVTYPE macro instruction to request information relating to the characteristics of an I/O device, and to cause this information to be placed into a specified area. (The results of a DEVTYPE macro instruction executed before a checkpoint is taken should not be considered valid after a checkpoint/restart occurs.)

The topics that follow discuss the macro itself, device characteristics, and particular output for particular devices.

DEVTYPE Macro Specification

The format is:

<i>[symbol]</i>	DEVTYPE	<i>ddloc-addrx,area-addrx[,DEVTAB][,RPS]</i>
-----------------	----------------	----------------------------------------------

ddloc-addrx

the address of an 8-byte field that contains the symbolic name of the DD statement to which the device is assigned. The name must be left justified in the 8-byte field, and must be followed by blanks if the name is less than eight characters. The doubleword need not be on a doubleword boundary.

area-addrx

the address of an area into which the device information is to be placed. The area can be one, two, five, or six fullwords, depending on whether or not the DEVTAB and RPS operands are specified. The area must be on a fullword boundary.

DEVTAB

This operand is only required for direct-access devices. If DEVTAB is specified, the following number of words of information is placed in your area:

- For direct-access devices - 5 words
- For non-direct-access devices - 2 words

If you do not code DEVTAB, one word of information is placed in your area if the reference is to a graphics or teleprocessing devices; for any other type of device, two words of information are placed in your area.

RPS

If RPS is specified, DEVTAB must also be specified. The RPS parameter causes one additional full word of RPS information to be included with the DEVTAB information.

Note: Any reference to a dummy DD statement in the DEVTYPE macro instruction will cause eight bytes of zeros to be placed in the output area. For installations using OS/VS1, any reference to a JES spool device causes zeros to be placed in word 1 and 32,767 to be placed in word 2 in the output area.

Device Characteristics Information

The following information is placed into your area as a result of issuing a DEVTYPE macro:

Word 1

Describes the device as defined in the UCBTYP field of the UCB. For a complete description of this field, refer to:

- *OS/VS1 System Data Areas*, if you're using OS/VS1
- *OS/VS2 System Data Areas*, if you're using OS/VS2.

Word 2

Maximum blocksize. For direct-access devices, this value is the maximum size of an unkeyed block; for magnetic or paper tape devices, this value is the maximum blocksize allowed by the operating system. For all other devices, this value is the maximum blocksize accepted by the device.

If DEVTAB is specified, the next three fullwords contain the following information about direct-access devices:

Word 3

Bytes 1-2	The number of physical cylinders on the device.
Bytes 3-4	The number of tracks per cylinder.

Word 4

Bytes 1-2	Maximum track length. Note that for the 2305 and 3330 direct-access devices, this value is not equal to the value in word two (maximum blocksize) as it is for other IBM direct-access devices.
Byte 3	Block overhead, keyed block – the number of bytes required for gaps and check bits for each keyed block other than the last block on a track.
Byte 4	Block overhead – the number of bytes required for gaps and check bits for a keyed block that is the last block on a track.

Word 5

Byte 1	Block overhead, block without key – the number of bytes to be subtracted if a block is not keyed.
Byte 2	bits 0-3 Reserved. bit 4 If 1, bytes 3 and 4 of word 4 contain a halfword giving the block overhead for any block on a track, including the last block. bits 5-6 Reserved. bit 7 If 1, a tolerance factor must be applied to all blocks except the last block on the track.
Bytes 3-4	Tolerance factor - this factor is used to calculate the effective length of a block. The calculation should be performed as follows: <i>Step 1</i> - add the block's key length to the block's data length. <i>Step 2</i> - test bit 7 of byte 2 of word 5. If bit 7 is 0, perform step 3. If bit 7 is 1, multiply the sum computed in step 1 by the tolerance factor. Shift the result of the multiplication lines bits to the right. <i>Step 3</i> - add the appropriate block overhead to the value obtained above.

If DEVTAB and RPS are specified, the next fullword contains the following information:

Word 6

Bytes 1-2	R0 overhead for sector calculations
Byte 3	Number of sectors for the device
Byte 4	Number of data sectors for the device

Output for Each Device Type

Device	UCB Type Field (Word 1)**	Maximum Record Size (Word 2, In Decimal)	DEVTAB (Words 3, 4, and 5, In Hexadecimal)	RPS (Word 6, (In Hexadecimal)
2540 Reader		80	Not Applicable	Not Applicable
2540 Reader w/CI		80	Not Applicable	Not Applicable
2540 Punch		80	Not Applicable	Not Applicable
2540 Punch w/CI		80	Not Applicable	Not Applicable
1442 Reader-Punch		80	Not Applicable	Not Applicable
1442 Reader-Punch w/CI		80	Not Applicable	Not Applicable
1442 Serial Punch		80	Not Applicable	Not Applicable
1442 Serial Punch w/CI		80	Not Applicable	Not Applicable
2501 Reader		80	Not Applicable	Not Applicable
2501 Reader w/CI		80	Not Applicable	Not Applicable
2520 Reader-Punch		80	Not Applicable	Not Applicable
2520 Reader-Punch w/CI		80	Not Applicable	Not Applicable
2520 B2-B3		80	Not Applicable	Not Applicable
2520 B2-B3 w/CI		80	Not Applicable	Not Applicable

Device	UCB Type Field (Word 1)**	Maximum Record Size (Word 2, In Decimal)	DEV TAB (Words 3, 4, and 5, In Hexadecimal)	RPS (Word 6, In Hexadecimal)
1287 Optical Reader		0	Not Applicable	Not Applicable
1288 Optical Reader		0	Not Applicable	Not Applicable
1419/1275 Reader/Sorter		0	Not Applicable	Not Applicable
3505 Reader		80	Not Applicable	Not Applicable
3505 Reader w/CI		80	Not Applicable	Not Applicable
3525 Punch		80	Not Applicable	Not Applicable
3525 Punch w/CI		80	Not Applicable	Not Applicable
1403 Printer		120*	Not Applicable	Not Applicable
1403 w/UCS		120*	Not Applicable	Not Applicable
1404 Printer		120*	Not Applicable	Not Applicable
1443 Printer		120*	Not Applicable	Not Applicable
3211 Printer		132*	Not Applicable	Not Applicable
2671 Paper Tape Reader		32767	Not Applicable	Not Applicable
1052 Printer-Keyboard		130	Not Applicable	Not Applicable
1053 Printer			Not Applicable	Not Applicable
3210 Printer-Keyboard		130	Not Applicable	Not Applicable
3215 Printer-Keyboard		130	Not Applicable	Not Applicable
2400 (9-track)		32767	Not Applicable	Not Applicable
2400 (9-track, p.e.)		32767	Not Applicable	Not Applicable
2400 (9-track, d.d.)		32767	Not Applicable	Not Applicable
2400 (7-track)		32767	Not Applicable	Not Applicable
2400 (7-track, d.c.)		32767	Not Applicable	Not Applicable
2495 Tape Cartridge Reader		0	Not Applicable	Not Applicable
3400 (9-track, p.e.)		32767	Not Applicable	Not Applicable
3400 (9-track, d.d.)		32767	Not Applicable	Not Applicable
3400 (7 track)		32767	Not Applicable	Not Applicable
2314/2319 DAS Facility		7294	00CB00141C7E922D2D010216	Not Applicable
2305-1 Fixed-Head Storage		14138	0030000838E80278CA090200	02985A57
2305-2 Fixed-Head Storage		14660	006A00083A0A01215B090200	0140B4B1
3330 Disk Storage		13030	019B0013336DBFBF38010200	00ED807C
2250-1 Display Unit			Not Applicable	Not Applicable
2250-2 Display Unit			Not Applicable	Not Applicable
2253-3 Display Unit			Not Applicable	Not Applicable

Legend

CI-card image feature, d.c.-data conversion, d.d.-dual density, p.e.-phase encoding, UCS-universal character set, w/-with

*Although certain models can have a larger line size, the minimum line size is assumed.

**Device codes are presented in the data areas publications:

For VS1 systems, see **OS/VS1 System Data Areas**, "The UCBTYP Field of the UCB."

For VS2 systems, see **OS/VS2 System Data Areas**, "The UCBTYP Field of the UCB."

Communication Equipment

1030,1050,83B3, TWX,2250, S360
1060,115A,1130
2780
2740

Record Size

Not Applicable
Not Applicable
Not Applicable
Not Applicable

Control is returned to your program at the next executable instruction following the DEVTYPE macro instruction. If the information concerning the DDNAME you specified has been successfully moved to your work area, register 15 will contain zeros. Otherwise, register 15 will contain one of the following exception return codes.

- 04 - DDname not found.
- 08 - Invalid area address. The address of the output area either violates protection, or it is out of the range of virtual storage.

Reading and Modifying a Job File Control Block

To accomplish the functions that are performed as a result of an OPEN macro instruction, the Open routine requires access to information that you have supplied in a data definition (DD) statement. This information is stored by the system in a job file control block (JFCB).

Usually, the programmer is not concerned with the JFCB itself. In special applications, however, you may find it necessary to modify the contents of a JFCB before issuing an OPEN macro instruction. To assist you, the system provides the RDJFCB macro instruction. This macro instruction causes a specified JFCB to be read into virtual storage from the job queue (SYS1.SYSJOBQE) or system work area data set in which it has been stored. The format and field descriptions of the JFCB are contained in *OS/VS1 System Data Areas* for VS1 systems and in *OS/VS2 System Data Areas* for VS2 systems.

When subsequently issuing the OPEN macro instruction, you must indicate, by specifying the TYPE=J option, that you have supplied a modified JFCB to be used during the initialization process.

The JFCB is returned to SYS1.SYSJOBQE or the system work area data set by the Open routine or the OPENJ routine, if any of the modifications in the following list occur. These modifications can occur only if the information is not in the JFCB when the OPEN macro instruction is issued.

- Expiration date field and creation date field merged into the JFCB from the DSCB.
- Secondary quantity field merged into the JFCB from the DSCB.
- DCB fields merged into the JFCB from the DSCB.
- DCB fields merged into the JFCB from the DCB.
- Volume serial number fields added to the JFCB.
- Data set sequence number field added to the JFCB.
- Number of volumes field added to the JFCB.

If you make these, or any other modifications, and you want the JFCB returned to the job queue or system work area data set, you must set the high-order bit of field JFCBMASK+4 to one. This field is in the JFCB. Setting the high-order bit of field JFCBMASK+4 to zero does not necessarily suppress the return of the JFCB to the job queue or system work area data set. If the OPEN or OPENJ routines have made any of the above modifications, the JFCB is returned to the job queue or system work area data set. To inhibit writing the JFCB back to the job queue or system work area data set during an OPENJ, the field JFCBTSDM should be set to X'08' prior to issuing the OPEN macro.

Examples: The macro instruction at EX1 creates a parameter list for two data control blocks: INVEN and MASTER. In creating the list, both data control blocks are assumed to be opened for input; option₂ for both blocks is assumed to be DISP. The macro instruction at EX2 reads the system-created JFCBs for INVEN and MASTER from the job queue into virtual storage, thus making the JFCBs available to the problem program for modification. The macro instruction at EX3 modifies the parameter list entry for the data control block named INVEN and indicates, through the TYPE=J operand, that the problem program is supplying the JFCBs for system use.

```

EX1      RDJFCB ( INVEN , , MASTER ) , MF=L
        .
        .
EX2      RDJFCB MF=( E , EX1 )
        .
        .
EX3      OPEN ( , ( RDBACK , LEAVE ) ) , TYPE=J , MF=( E , EX1 )
        .
        .
INVEN   DCB      EXLST=LSTA , ...
MASTER  DCB      EXLST=LSTB , ...
LSTA    DS      0F
        DC      X'07'
        DC      AL3( JFCBAREA )
        .
        .
JFCBAREA DS      176C
        .
        .
LSTB    DS      0F
        .
        .

```

Any number of data control block addresses and associated options may be specified in the RDJFCB macro instruction. This facility makes it possible to read job file control blocks in parallel.

An exit list address must be provided in each data control block specified by an RDJFCB macro instruction. Each exit list must contain an active entry that specifies the virtual storage address of the area into which a JFCB is to be placed. A full discussion of the exit list and its use is contained in *OS/VS Data Management Services Guide*. The format of the job file control block exit list entry is as follows:

Type of Exit List Entry	Hexadecimal Code (high-order byte)	Contents of Exit List Entry (the low-order bytes)
Job file control block	07	Address of a 176-byte area to be provided if the RDJFCB or OPEN (TYPE=J) macro instruction is used. This area must begin on a fullword boundary and must be located within the user's region.

The virtual storage area into which the JFCB is read must be at least 176 bytes long.

The data control block may be open or closed when this macro instruction is executed.

If the JFCB is read successfully for all DCBs in the parameter list, a return code of zero is placed in register 15. If the JFCB is not read for any of the DCBs because the DDNAME is blank or a DD statement is not provided, then a return code of 4 is placed in register 15.

Cautions: The following errors cause the results indicated:

Error	Result
A DD control statement has not been provided.	A return code of 4 is placed in register 15.
DDNAME field in DCB is blank.	A write-to-programmer is issued, the request for this DCB is ignored, and a return code of 4 is placed in register 15.
A virtual storage address has not been provided.	Abnormal termination of task.

Note that if you want to open a VTOC data set to change its contents (that is, open it for OUTPUT, OUTIN, INOUT, or UPDAT), your program must be authorized under the Authorized Program Facility (APF). APF provides security and integrity for your data sets and programs. Details on how you authorize your program are provided in the *OS/VS2 Planning and Use Guide*. APF information for installations using OS/VS1 is provided here for planning purposes only.

Ensuring Data Security by Validating the Data Extent Block

Protecting one user's data from inadvertent or malicious access by an unauthorized user depends on protection of the data extent block (DEB). The DEB is a critical control block because it contains information about the device a data set is mounted on and describes the location of data sets on direct-access device storage volumes. The DEB also contains the address of the appendage vector table (AVT). Using the AVT, a user with malicious intent can modify the AVT to give control to his own routine in supervisor state to read from and write to data sets to which he would otherwise be denied access.

To guarantee protection of the DEB, the DEBCHK macro instruction is provided. The DEBCHK macro is issued by several components of the system control program. For example,

- the Open access method executors issue the macro to add the address of a DEB they have built to a list of valid addresses called the *DEB table*. The DEB validity checking routine builds and maintains a DEB table for each job step.
- the I/O supervisor uses the macro to verify that the DEB passed with each EXCP request is in the DEB table.
- the Close component issues the macro to remove a DEB from the DEB table .

If you code a routine that builds a DEB, you must add the address of the DEB you built to the DEB table. If you code a routine that depends on the validity of a DEB that is passed to your routine, you should verify that the DEB passed to your routine has a valid entry in the DEB table. Use the **TYPE=ADD** and the **TYPE=VERIFY** operands of the macro, respectively.

Additional details about the functions provided by the DEB validity checking routine and about the contents of the DEB table are available in *OS/VS O/C/EOV Logic*, SY26-3785.

The DEBCHK macro instruction provides four functions:

- adds the address of a DEB to the DEB table, which is located in protected storage. The DEB table contains the address of every user DEB associated with a given job step. Every system control program component that builds a user DEB must add the address of that DEB to a DEB table.
- verifies that the DEB table associated with a given job step contains the address of a valid DEB. Any system control program component or problem program can use this function to verify that a DEB is valid.
- deletes the address of a DEB from the DEB table. Any program that deletes a user DEB must, before it deletes the DEB, issue a DEBCHK macro with a **TYPE=DELETE** operand to delete the address of the DEB from the DEB table. If the DEB validity checking routine encounters an error while deleting the address from the DEB table, the job step is abnormally terminated.
- deletes the address of a DEB from the DEB table in the same way as the preceding function, except that, instead of terminating the job step, this function merely returns an error code in register 15. This function is provided to prevent recurring abnormal termination. The format of the DEBCHK and a description of the operands follow:

DEBCHK – Macro Specification

<i>[symbol]</i>	DEBCHK	<i>cbaddr</i> [,TYPE =	{ VERIFY }	{ amtype })
		(,MF = L	{ ADD }	[,AM = {(amaddr)}]
			{ DELETE }	{((amreg))}]
			{ PURGE }	

cbaddr RX-Type Address, (2-12), or (1)
 a control block address passed to the DEBCHK routine. This operand is ignored if MF=L is coded. For verify, add, and delete requests, *cbaddr* is the address of a data control block (DCB) that points to the DEB whose address is either verified to be in the DEB table, added to the DEB table, or deleted from the DEB table. For the purge function, *cbaddr* is the address of the DEB whose pointer is to be purged from the table: no reference is made to the DCB.

```

      {VERIFY }
TYPE= {ADD }
      {DELETE }
      {PURGE }
  
```

indicates the function to be performed. If MF=L is coded, TYPE is ignored. The functions are:

ADD. Before the DEB pointer can be added to the table, the DEB must be queued on the current TCB DEB chain (the TCBDEB field contains the address of the first DEB in the chain). The DEB address is added to the DEB table at some offset into the table. That offset value is placed in the DEBTBLOF field of the DEB, and the access method type is inserted into the DEBAMTYP field of the DEB. A zero is placed in the DEBAMTYP field if the AM operand is not coded. **TYPE=ADD** can be issued only in supervisor state.

VERIFY. This function is assumed if the **TYPE** operand is not coded. The control program checks the DEB table to determine whether the DEB pointer is in the table at the location indicated by the DEBTBLOF field of the DEB; the DEBAMTYP field in the DEB is compared to the AM operand value, if given. The two must be equal. **TYPE=VERIFY** can be issued in either supervisor or problem state.

DELETE. The DEB and the DCB must point to each other before the DEB address can be deleted from the DEB table. **TYPE=DELETE** can be issued only in supervisor state.

PURGE. The DEB pointer is removed from the DEB table without checking the DCB. **TYPE=PURGE** can be issued only in supervisor state.

AM

specifies an access method value. Each value corresponds to a particular access method type (note that BPAM and SAM have the same values):

Type	Value
ISAM	X'80'
BDAM	X'40'
SAM	X'20'
BPAM	X'20'
TAM	X'10'
GAM	X'08'
TCAM	X'04'
EXCP	X'02'
NONE	X'00'

The operand can be coded in one of the following three ways, only the first of which is valid for the list form (**MF=L**) of the instruction.

amtype

refers to the actual access method type: **ISAM**, **BDAM**, **SAM**, **BPAM**, **TAM**, **GAM**, **TCAM**, or **EXCP**.

amaddr

is the RS-type address of the access method value. This format may not be coded when **MF=L** is used.

amreg

is one of the general registers 1-14 that contains the access method value in its low-order byte (bit positions 24-31). The high-order bytes are not inspected. This form may not be used when **MF=L** is coded.

The use of *amaddr* and *amreg* should be restricted to those cases where the access method value has been generated previously by the **MF=L** form of DEBCHK. If **MF=L** is not coded, the significance of the AM operand depends upon the **TYPE**.

If **TYPE** is **ADD** and **AM** is specified, the access method value is inserted in the DEBAMTYP field of the DEB, and all subsequent DEBCHK macros referring to this DEB must either specify the same **AM** or omit the operand. When the **AM** operand is omitted for **TYPE=ADD**, a null value (0) is placed in the DEB and all subsequent DEBCHK macros must omit the **AM operand**.

If **AM** is specified when the **TYPE** is **PURGE**, **DELETE**, or **VERIFY**, the access method value is compared to the value in the DEBAMTYP field of the DEB. If **AM** is omitted, no comparison is made.

MF

indicates the list form of the DEBCHK macro instruction. When MF=L is coded, a parameter list is built consisting of the access method value that corresponds to the AM keyword. This value may be referenced by name in another DEBCHK macro by coding AM= (*amaddr*), or it may be inserted into the low-order byte of a register before issuing another DEBCHK macro by coding AM= (*(amreg)*).

Removing Queued Requests and Restoring the Requests

You can stop the processing of I/O requests for a specific task or against a particular data set, using the PURGE macro instruction. The function of the PURGE macro instruction is to call the PURGE routine which removes request queue elements (RQEs) from queues and frees RQEs. You can subsequently requeue the requests by issuing the RESTORE macro. The PURGE and RESTORE macros give control to routines documented in *OS/VS I/O Supervisor Logic, SY24-5156*. The logic of the routines and additional details are available in that publication.

You can give control to the Purge and Restore routines in two ways: (1) by loading register 1 with the address of your parameter list and issuing the assembler language SVC instructions or (2) by issuing the PURGE and RESTORE macro instructions. If your installation requires the use of macro instructions, you must add the macro definitions to the macro library (SYS1.MACLIB) or place them in a partitioned data set and concatenate this data set to the macro library. The macro definitions, JCL, and utility statements needed to add the macros to your macro library are presented in Figures 24 and 25. Whether you issue the macro instructions or the SVC instructions, you must first build a parameter list. The SVC instructions are SVC 16 for PURGE and SVC 17 for RESTORE.

PURGE Macro Definition

```
MACRO
&NAME PURGE      &LIST
AIF    ('&LIST'EQ''').E1
&NAME IHBINNRA  &LIST      LOAD REG 1
SVC    16
MEXIT
.E1    IHBERMAC  01,147      LIST ADDR MISSING
MEND
```

Control Statements Required

```
//jobname      JOB          {parameter}
//stepname     EXEC        PGM=IEBUPDTE,PARM=NEW
//SYSPRINT     DD          SYSOUT=A
//SYSUT2       DD          DSN=SYS1.MACLIB,DISP=OLD
//SYSIN        DD          *
./  ADD        NAME=PURGE,LIST=ALL
.
.
.
PURGE macro definition
.
.
./  ENDUP
/*
```

Figure 24. Macro Definition, JCL, and Utility Statements for Adding PURGE Macro to Your Macro Library

```

RESTORE Macro Definition

&NAME      MACRO
RESTORE    RESTORE    &LIST
AIF        ('&LIST' EQ '').E1
&NAME      IHBINNRA  \&LIST      LOAD REG 1
SVC        17          ISSJUE SVC FOR RESTORE
MEXIT
.E1        IHERMAC   01,150      LIST ADDR MISSING
MEND

```

Control Statements Required

```

//jobname   JOB          {parameters}
//stepname  EXEC        PGM=IEBUPDTE, PARM=NEW
//SYSPRINT  DD          SYSOUT=A
//SYSUT2    DD          DSNAME=SYS1.MACLIB, DISP=OLD
//SYSIN     DD          DATA
./  ADD     NAME=RESTORE, LIST=ALL
.
.
.
RESTORE macro definition
.
.
./  ENDUP
/*

```

Figure 25. Macro Definition, JCL, and Utility Statements for Adding RESTORE Macro to Your Macro Library

PURGE – Remove an RQE From a Queue

The Purge routine stops the processing of I/O requests by removing RQEs from queues. The queues from which RQEs can be removed are:

- a logical channel queue
- the task supervisor request block queues
- the task supervisor asynchronous exit queue
- the dynamic device reconfiguration (DDR) WAIT queue

The macro instruction used to call the Purge routine is coded as follows:

```

[symbol]      PURGE      parameter-list address

```

parameter list address, RX-type address, (2-12) or (1) specifies the address of a parameter list you have built on a fullword boundary in your region. The parameter list address can be specified as an RX-type constant or in registers 2-12 or 1. You specify which queues you want altered by bit setting in the first field (PRGOPT) of the parameter list. You also can choose to either halt any currently active I/O operation (requests cannot be restored) or allow the operation to quiesce (requests can be restored), again by using a bit setting in the PRGOPT field of the parameter list (see Figure 26).

Note that you can bypass the purge of the request blocks chained to a TCB (or to be chained to a TCB) by setting bit 5 of the PRGOPT field to 1.

0(0) PRGOPT Purge Options	1(1) PRGDEB Address of Data Extent Block
4(4) PRGCOD Complete Code	5(5) PRGTCB Address of Task Control Block
8(8) PRGCTR Quiesce Count	9(9) PRGCHN Address of First Link in Chain

Figure 26. PURGE Parameter List

Offset	Bytes and Alignment	Field Name	Description
0 (0)	1	PRGOPT	Purge options.
		0... ..	Purge request queue elements for all entries in the data extent block (DEB) chain, starting with the DEB whose address is in PRGDEB.
		1... ..	Purge only the request queue element for the DEB whose address is in PRGDEB.
		.0.. ..	Do not post the event control blocks for the purged request queue elements.
		.1.. ..	Post the event control blocks for the purged request queue elements. (A X'48' completion code is used.)
		..0.	Allow the activity to quiesce.
		..1.	Halt the I/O activity. (The effect of the Halt I/O instruction is simulated if the operation is a seek.)
		...0	Purge all requests.
		...1	Purge only related requests.
	 0...	Normal purge.
	 1...	Used only for TSO tasks.
	0..	Purge the asynchronous exit queue, the request block queue, the logical channel queue, and the DDR wait queue.
	1..	Purge the logical channel queue, the asynchronous exit queue (removing only RQEs for requests in error), and the DDR wait queue. Bypass the request blocks.
	0.	Purge by data extent block.
	1.	Purge by task control block. When this bit is on, the setting of bit 0 is ignored.
	x	Reserved.
1 (1)	. 3	PRGDEB	Address of data extent block. If you are purging by TCB, not required.
4 (4)	1	PRGCOD	Completion code.

Offset	Bytes and Alignment	Field Name	Description
5 (5)	. 3	PRGTCB	Address of task control block. If none, the current TCB is used.
8 (8)	1	PRGCTR	Quiesce count. The number of active request queue elements for which I/O activity has not yet been completed.
9 (9)	. 3	PRGCHN	Address of the first link in the chain of IOBs which are purged. The first link can be located in the user's area, or in the DEBUSPRG field of the DEB. It will point to the first IOB in the chain. The last IOB in the chain will contain ones in the low-order byte of the restart address (IOBR) field. A diagram of the purge chain is shown in Figure 27.

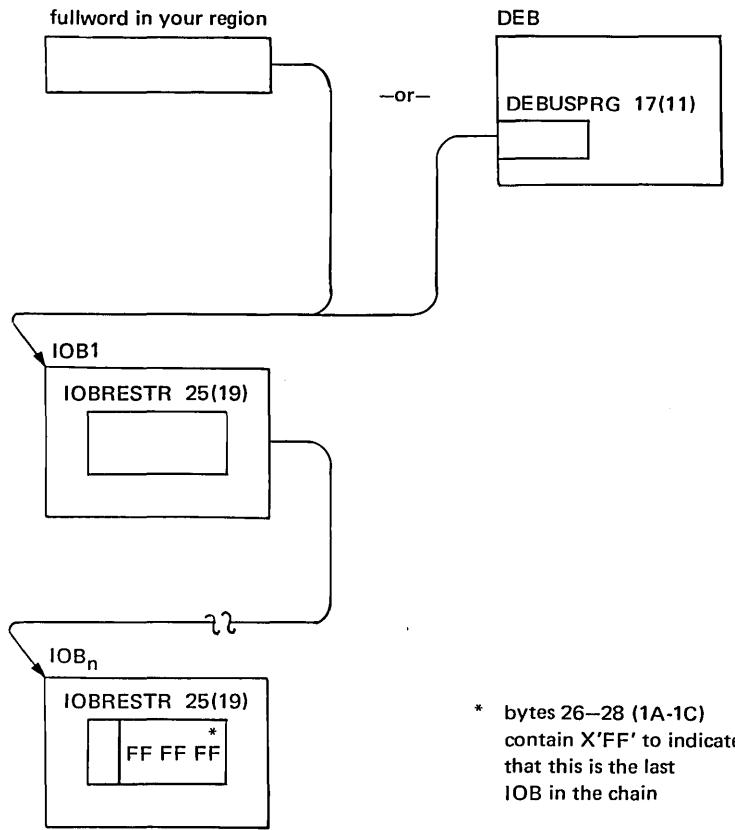
If you are purging all the I/O requests currently in a queue for a given data set using the quiesce option, a chain of IOBs will be built whose addresses represent RQEs that have been removed from a queue. When control is returned to your program, the address of a pointer to the first IOB that was dequeued will be in the PRGCHN field of your parameter list. This address is used to restore the requests to queues.

RESTORE – Return Purged IOBs to Queues

You can restore I/O requests to the queues from which they were purged by issuing the RESTORE macro instruction, which can be coded as follows:

```
[symbol]      RESTORE      purge chain-address
```

purge chain-address – RX-type address, (2-12) or (1)
specifies the address of the first of one or more IOBs you want restored to queues. The *purge chain address* may be specified as either an RX-type constant or loaded into registers 2-12 or 1. This field can be either (1) the address of the DEBUSPRG field (offset 17 (X'11')) in a DEB or (2) a fullword in your region. The IOBRESTR field (offset 24 (X'18')) of the IOB is used to chain IOBs. The last three bytes of the IOBRESTR field of the last IOB in the chain are set to X'FF' (see Figure 27).



* bytes 26–28 (1A-1C) contain X'FF' to indicate that this is the last IOB in the chain

Figure 27. Purge Chain for Restoring IOBs

ADDING A UCS IMAGE OR FCB IMAGE TO THE SYSTEM IMAGE LIBRARY

This chapter provides a detailed description of how to add either an IBM UCS (universal character set) image or an IBM FCB (forms control buffer) image to SYS1.IMAGELIB.

Before reading this section, you should be familiar with the information in these publications:

- *IBM 2821 Control Unit, GA24-3312*, contains the information necessary to create a user-designed chain/train for the 1403 Printer.
- *OS/VS Data Management Macro Instructions, GC26-3793*, describes the SETPRT macro instruction that loads a UCS image and an FCB image into their respective buffers.
- *OS/VS JCL Reference, GC28-0618*, describes the UCB and FCB parameters that can be specified in a DD statement to load the UCS and FCB buffers when they are opened.
- *IBM 3211 Printer and 3811 Control Unit Component Description, GA24-3543*, contains the information necessary to create a user-designed train for the 3211 Printer.

Introduction

Unlike most other chapters in this publication, this chapter does not explain macro instructions. Rather, it shows how you can use the assembler and linkage editor to place an image in the library; no executable code is generated – the assembler prepares DCs and the linkage editor puts them in the library.

The remainder of this chapter discusses, first, UCS images, and, then, FCB images.

Adding a UCS Image to the Image Library

The IBM standard character set images listed in the following table may be included in SYS1.IMAGELIB at system generation by using the UCS macro instruction. You code a member name for an image in the image library by prefixing a character set code shown in the table with UCS1 or UCS2. UCS1 denotes a 1403 printer image and UCS2 denotes a 3211 printer image (for example, UCS1AN or UCS2A11).

1403	AN, HN, PCAN, PCHN, PN, QNC, QN, RN, SN, TN, XN, YN
3211	A11, G11, H11, P11, T11

You may add a user-designed character image to the image library or make an existing image a default image by following these rules:

1. The member name must be either the four characters UCS1 for the 1403 or UCS2 for the 3211 printer. The member name must be followed by a unique character set code that is one to four characters long. This character set code can be any valid combination of letters and numbers according to the rules for assembler language symbols. The single letters U or C should not be used as a character set code since they are symbols for special conditions recognized by the system. The assigned

character set code must be specified on the DD statement or SETPRT macro instruction to load the image into the UCS buffer.

2. The first byte in the load module of a character set image specifies whether or not the image is a default. A default image is indicated by X'80', and is used when the UCS parameter is not coded in the DD statement. X'00' specifies that the image is not to be used as a default.
3. The second byte of the load module indicates the number of lines (n) to be printed for image verification.
4. Each byte of the next n bytes indicates the number of characters to be printed on each verification line. (Note: For the 3211 printer, the maximum number of characters printed per line is 48; the associative bytes are not printed during verification.)
5. A 240-byte 1403 UCS image or a 512-byte 3211 UCS image must follow the previously described fields. (A 3211 UCS image has 432 characters, followed by 15 bytes of X...00', 64 bytes of associative bits, and a reserved byte (byte 512) of X'00'.) Two apostrophes or two ampersands must be coded to represent a single apostrophe or a single ampersand, respectively, which is a part of a character set image.

The following code is an example of adding a 1403 UCS image, YN, to the image library.

```

//ADDYN      JOB MSGLEVEL=1
//STEP       EXEC PROC=ASMFCL, PARM.ASM='NODECK,LOAD',           X
//           PARM.LKED='LIST,NCAL,NE,OL'
//ASM.SYSIN  DD *
UCS1YN      CSECT
            DC X'80'          (THIS IS A DEFAULT IMAGE)
            DC AL1(6)         (NUMBER OF LINES TO BE PRINTED)
            DC AL1(39)        (39 CHARACTERS PRINTED ON 1ST LINE)
            DC AL1(42)        (42 CHARACTERS PRINTED ON 2ND LINE)
            DC AL1(39)        (39 CHARACTERS PRINTED ON 3RD LINE)
            DC AL1(39)        (39 CHARACTERS PRINTED ON 4TH LINE)
            DC AL1(42)        (42 CHARACTERS PRINTED ON 5TH LINE)
            DC AL1(39)        (39 CHARACTERS PRINTED ON 6TH LINE)
            DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*..'
            DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*..#-$'
            DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*..'
            DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*..'
            DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*..#-$'
            DC C'1234567890STABCDEFGHIJKLMNOPQRSTUVWXYZ*..'
            END
/*
//LKED.SYSLMOD DD DSN=SYS1.IMAGELIB(UCS1YN),DISP=OLD

```

The following example shows the code used to add a 3211 UCS image (A11) to the image library. The first 432 bytes of the 3211 UCS image correspond to the 432 positions on the print train. The next 64 bytes are associative bits used to avoid data checks. See the publication *IBM 3211 Printer and 3811 Control Unit Component Description*, GA24-3543, to determine how to code these bits for a particular train.

Note: Executing the ASMFCL procedure does not actually generate executable code. The assembler/linkage editor is used as a vehicle to load the UCS image into the image library.

```

//ADDA11      JOB MSGLEVEL=1
//STEP        EXEC PROC=ASMFCL, PARM.ASM='NODECK, LOAD',
//            PARM.LKED='LIST, NCAL, NE, OL'
//ASM.SYSIN   DD *
UCS2A11      CSECT
              DC X'80'          (THIS IS A DEFAULT IMAGE)
              DC AL1(9)         (NUMBER OF LINES TO BE PRINTED)
              DC AL1(48)        (48 CHARACTERS PRINTED ON 1ST LINE)
              DC AL1(48)        (48 CHARACTERS PRINTED ON 2ND LINE)
              DC AL1(48)        (48 CHARACTERS PRINTED ON 3RD LINE)
              DC AL1(48)        (48 CHARACTERS PRINTED ON 4TH LINE)
              DC AL1(48)        (48 CHARACTERS PRINTED ON 5TH LINE)
              DC AL1(48)        (48 CHARACTERS PRINTED ON 6TH LINE)
              DC AL1(48)        (48 CHARACTERS PRINTED ON 7TH LINE)
              DC AL1(48)        (48 CHARACTERS PRINTED ON 8TH LINE)
              DC AL1(48)        (48 CHARACTERS PRINTED ON 9TH LINE)
*
*            THE FOLLOWING NINE LINES REPRESENT
*            THE TRAIN IMAGE
              DC C'1234567890#@/STUVWXYZ&&&, %JKLMNOPQR-$*ABCDEFGHI=. '
              DC C'1234567890#@/STUVWXYZ&&&, %JKLMNOPQR-$*ABCDEFGHI=. '
              DC C'1234567890#@/STUVWXYZ&&&, %JKLMNOPQR-$*ABCDEFGHI=. '
              DC C'1234567890#@/STUVWXYZ&&&, %JKLMNOPQR-$*ABCDEFGHI=. '
              DC C'1234567890#@/STUVWXYZ&&&, %JKLMNOPQR-$*ABCDEFGHI=. '
              DC C'1234567890#@/STUVWXYZ&&&, %JKLMNOPQR-$*ABCDEFGHI=. '
              DC C'1234567890#@/STUVWXYZ&&&, %JKLMNOPQR-$*ABCDEFGHI=. '
              DC C'1234567890#@/STUVWXYZ&&&, %JKLMNOPQR-$*ABCDEFGHI=. '
              DC C'1234567890#@/STUVWXYZ&&&, %JKLMNOPQR-$*ABCDEFGHI=. '
              DC 15X'00'        RESERVED FIELD, BYTES 433-447
* THE FOLLOWING FOUR DC INSTRUCTIONS DEFINE THE ASSOCIATIVE BITS,
* UCSB BYTE POSITIONS 448-511
              DC X'CO1010101010101010100040404240004010'
              DC X'10101010101010101010004041000040401010'
              DC X'101010101010004040000000101010101010'
              DC X'10101010004040444800'
              DC X'00'          RESERVED FIELD, BYTE 512
              END
/*
//LKED.SYSLMOD DD      DSNAME=SYS1.IMAGELIB(UCS2A11), DISP=OLD

```

Adding an FCB Image to the Image Library

Two standard FCB images, STD1 and STD2, can be included in SYS1.IMAGELIB during system generation for a 3211 printer. STD1 prints six lines per inch on a 8 1/2 inch form. STD2 prints six lines per inch on an eleven inch form. Channels for both images are evenly spaced with channel one on the fourth line and channel nine on the last line.

In addition to the IBM-supplied images, user images can be defined. Each user image is added to the image library as part of a load module. To add an FCB image to the image library, follow these rules:

1. The member name cannot exceed eight bytes. The first four characters of this member name must be FCB2. The characters that follow FCB2 identify the FCB image and are referred to as the image identifier. Any combination of characters that are valid in assembler language can be used with the exception of a single "S" or a single "U" as an image identifier. The image identifier must be specified in a DD statement or in the SETPRT macro instruction to load the image in the FCB buffer.
2. The first byte of the load module of a forms control image specifies whether or not the image is a default. A default image is indicated by X'80' and is used for all jobs

that do not have the FCB parameter coded on the DD statement; X'00' indicates that the image is not to be used as a default.

3. The second byte of the load module indicates the number of lines per form (FCB image length). The maximum image length is 180 lines. The FCB image must be as long as the form. For example, if you are printing eight lines per inch on an eleven inch form, the FCB image must be 88 bytes long; if you are printing six lines per inch on the same form, the FCB image must be 66 bytes long.
4. The first byte of the FCB image (the third byte of the load module) defines the number of lines per inch and a channel:
 - X'1n' means eight lines are printed per inch.
 - X'0n' means six lines are printed per inch.

All remaining bytes (lines) must contain X'0n' except the last byte. The last byte must be X'1n'. The letter n can be a hexadecimal value from 1 to C, representing a channel (one to twelve); or it can be zero (0), which means no channel is indicated.

In the following example, an FCB load module is assembled and added to SYS1.IMAGELIB. The image defines a print density of eight lines per inch on an eleven inch form.

```

//ADDFCB      JOB      MSGLEVEL=1
//STEP        EXEC     PROC=ASMFEB, PARM.ASM='NODECK,LOAD'      X
//           PARM.LKED='LIST,NCAL,NE,OL'
//ASM.SYSIN   DD      *
FCB2ID1      CSECT
*THIS EXAMPLE IS FOR A FORM LENGTH OF 11 INCHES
*WITH 8 LINES OF PRINT PER INCH (88 LINES)
DC          X'80'      THIS IS A DEFAULT IMAGE
DC          AL1(88)    LENGTH OF FCB IMAGE
DC          X'10'      8 LINES PER INCH-NO CHANNEL FOR POS.1
DC          XL4'0'     4 LINES NO CHANNEL
DC          X'01'      CHANNEL 1 IN POSITION 6
DC          XL6'0'     6 LINES NO CHANNEL
DC          X'02'      CHANNEL 2 IN POSITION 13
DC          XL6'0'
DC          X'03'
DC          XL6'0'
DC          X'04'
DC          XL6'0'
DC          X'05'
DC          XL6'0'
DC          X'06'
DC          XL6'0'
DC          X'07'
DC          XL6'0'
DC          X'08'
DC          XL6'0'
DC          X'09'
DC          XL6'0'
DC          X'0A'
DC          XL6'0'
DC          X'0B'
DC          XL6'0'
DC          X'0C'      CHANNEL 12 IN POSITION 83
DC          XL4'0'     4 LINES NO CHANNEL
DC          X'10'      POSITION 88 - LAST LINE IN IMAGE
END
/*
//LKED.SYSLMOD DD      DSNAME=SYS1.IMAGELIB(FCB2ID1),DISP=OLD

```

INDEX

Indexes for reference manuals are consolidated in *OS/VS Master Index*, GC28-0602. For additional information about any subject listed below, refer to other publications listed for the same subject in the *Master Index*.



- abnormal end appendage (XCE) 57
- access method routines, functions performed in I/O operations 44
- alias name
 - assigning for an index (INDEX and CAMLST BLDA) 12-14
 - coding example 13
 - exceptional return codes 10
 - macro specifications 13
 - deleting for an index (INDEX and CAMLST DLTA) 13-14
 - coding example 14
 - exceptional return codes 10
 - macro specifications 13
 - entry in catalog 30
- alternate track 67
- APF (authorized program facility) 107
- appendages 50-58
 - naming convention 51
 - programming restrictions 51
- assigning an alternate track (ATLAS macro) 67
- ATLAS macro instruction (SVC 86) 67-72
 - coding example 69
 - how to use 69
 - operations performed 70
 - return codes 71
 - specification 68
 - with track overflow option 67
- authorized program facility (APF) 107



- BFALN operand of DCB macro 63
- BFTEK operand of DCB macro 63
- bit spinning on data read 59
- BLDA operand of INDEX CAMLST macro 13
- BLDG operand of INDEX CAMLST macro 10
- BLDX operand of INDEX CAMLST macro 13
- block multiplexer programming notes 58
- BUFCB operand of DCB macro 63
- BUFL operand of DCB macro 63
- BUFNO operand of DCB macro 63
- building a generation index (INDEX and CAMLST BLDG) 10
 - exceptional return codes 10
 - coding example 11
 - macro specifications 10
- building an index (INDEX and CAMLST BLDX) 9
 - coding example 9
 - exceptional return codes 10
 - macro specifications 9



- catalog block entries, data format 23-33
- CATALOG and CAMLST macro instructions 16-23
 - with CAT operand 17
 - with CATBX operand 19
 - with RECAT operand 22
 - with UCATDX operand 21
 - with UNCAT operand 20
- catalog maintenance
 - using CATALOG macro 16-23
 - using INDEX macro 9-16
 - using LOCATE macro 2-8
- cataloging data sets 16-20
 - when index levels exist (CATALOG and CAMLST CAT) 17
 - coding example 17
 - exceptional return codes 18
 - macro specifications 17
 - when index levels must be created (CATALOG and CAMLST CATBX) 18-20
 - coding example 19
 - exceptional return codes 18
 - macro specifications 19
- CCW (channel command word)
 - translation by I/O supervisor
 - for programs in nonpageable regions 45
 - for programs in pageable regions 53
 - normal 54
 - extended 53
 - (see also EXCP)
- CE (channel end)
 - appendage 56
 - operand of DCB 61
- channel program
 - appendages for use with 50
 - execution
 - (see EXCP)
 - initiation 47
 - related 49
 - translation
 - normal 49
 - extended 54
- checkpointed data sets, processing with EXCP 65
- checking the data extent block (DEB) 107
- CLOSE macro instruction
 - with EXCP 74
 - operations performed 74
 - options 74
 - specification 74
 - with XDAP 83
- CODE operand of DCB macro 65
- command retry for 3330 and 2305 58
- communication vector table (CVT) mapping macro 100

- completion code in ECB
 - with EXCP 78,49
 - with XDAP 83
- control password 93
- control block fields, EXCP 75
- control blocks
 - with EXCP 75
 - with XDAP 83
- control volumes
 - connecting (INDEX and CAMLST LNKX) 14-18
 - coding example 15
 - exceptional return codes 10
 - macro specifications 14
 - disconnecting (INDEX and CAMLST DRPX) 15-16
 - coding example 16
 - exceptional return codes 10
 - macro specifications 16
- control volumes (CVOL) pointer entry 29
 - before OS release 17 30
- conversion
 - relative block address to actual device address (IEPCNVT) 85
 - of sector value for RPS devices (IEC0SCR1) 87
- creating protected data sets 91
- CVOL
 - (see control volume)
- CVT (communication vector table) mapping macro 100



- DADSM routines 33
- data extent block (DCB)
 - fields with EXCP 77
 - validating 107
- data set catalogs 16-23
- data set control block
 - (see DSCB)
- data set pointer entry 23,26
- data set security
 - (see password protection and DEBCHK macro)
- DCB foundation block 117
- DCB, initialize for processing the JFCB 105
- DCB - define data control block for EXCP 60
- DCB
 - exit list entry for RDJFCB 106
 - space requirements 64
- DCBFDAD, maintaining 64
- DCBIFLGS field of DCB, permanent I/O error indicators 49
- DCBOFLGS field of DCB 73
 - bit settings for processing mod 73
- DCBTRBAL, maintaining 64
- DDR (dynamic device reconfiguration)
 - repositioning tape data sets 65
- DEB fields, EXCP 77
- DEBCHK macro instruction
 - functions of 108
 - specification 108
- defective track
 - (see ATLAS macro instruction) 67
- deleting a data set 36-38
 - coding example 37
 - exceptional return codes 38

- macro instructions for (SCRATCH and CAMLST SCRATCH) 37
 - with password protection 38
 - when volume not mounted 37
- deleting an index 11-13
 - coding example 12
 - exceptional return codes 10
 - macro specification for (INDEX and CAMLST DLTX) 12
- DEVD operand of DCB macro 64
- DEVD=DA, maintaining DCBFDAD and DCBTRBAL 64
- DEVD=TA, maintaining DCBBLKCT
 - for checkpointed data sets 65
 - for output data sets 64
 - for systems with DDR 65
- device characteristics 100
- device selection, EXCP 64
- device-dependent parameters, EXCP 64
- DEVTYPE macro instruction 100-104
 - output from 102
 - for RPS devices 101
 - specification 100
- direct-access device
 - channel program (XDAP macro) 80
- DLTA operand of INDEX CAMLST macros 13
- DLTX operand of INDEX CAMLST macros 12
- DRPX operand of INDEX CAMLST macros 16
 - space management (DADSM) routines 33
- DSCB, reading from VTOC (OBTAIN macro) 33-36
 - by data set name (SEARCH option) 34-35
 - coding example 34
 - exceptional return codes 35
 - macro specifications (OBTAIN and CAMLST SEARCH) 34
 - by relative block address (SEEK option) 35-36
 - coding example 35
 - exceptional return codes 36
 - macro specifications (OBTAIN and CAMLST SEEK) 35
- DSECT expansions
 - (see CVT, IEFJFCBN, IEFUCBOB)
- DSORG operand of DCB macro 63



- ECB fields
 - EXCP 77
 - XDAP 83
- end-of-extent appendage 56
- end-of-volume
 - condition 72
 - macro instruction (EOV) 74
 - on magnetic tape data sets 73
- EODAD operand of DCB macro 62
- EOV (end-of-volume) macro (SVC 55)
 - with EXCP 74
 - with XDAP 83
- error recovery procedures 51
- event control block fields
 - EXCP 77
 - XDAP 83
- exceptional return codes
 - ATLAS 71
 - CATALOG macro 18
 - INDEX macro 10

exceptional return codes (Continued)
 LOCATE macro 4
 OBTAIN macro 35,40
 RDJFCB macro 107
 RENAME macro 40
 SCRATCH macro 38
 EXCP (execute channel program) macro instruction
 command chaining 46
 control blocks used with
 DEB 77
 ECB 77
 IOB 75
 data chaining 46
 multivolume data set restriction 67
 in nonpageable partition 45
 other macros used with
 ATLAS 67
 CLOSE 74
 DCB 60
 EOVS 72
 OPEN 66
 in problem programs 45
 specification 67
 in system control programs 44
 validation of control blocks by I/O supervisor 47,107
 executing channel programs
 in problem programs 45
 in system control programs 44
 exit list entry for RDJFCB 106
 EXLST operand of DCB macro 62
 expiration date overriding 36

F

FCB (forms control buffer)
 format-1 DSCB, reading from VTOC 34
 forms control buffer (FCB) image
 adding image to SYS1.IMAGELIB 117
 rules 117
 foundation block of DCB
 defined 60
 extension 62
 parameters 61

G

generation data set, reading a catalog block for 4
 generation index pointer entry in catalog 31

I

I bit 54
 IECPCNVT (relative block to actual track address
 conversion routine) 85
 IEC0SCR1 (sector conversion routine) 87
 IEFJFCBN macro 100
 IEFUCBOB macro 99

IEHATLAS utility program 69
 IMSK operand of DCB macro 62
 INDEX and CAMLST macro instructions 9
 with BLDA operand 13
 with BLDG operand 10
 with BLDX operand 9
 with DLTA operand 13
 with DLTX operand 12
 with DRPX operand 16
 with LNKX operand 15
 index entries in catalog
 control entry 24
 link entry 25
 pointer entry 25
 index
 (see alias, assigning for an index; alias deleting
 for an index; building a generation index;
 building an index; deleting an index)
 input/output block fields
 EXCP 75
 XDAP 83
 interruption handling and error recovery procedures 49
 IOB fields
 EXCP 75
 XDAP 83
 IOBAD operand of DCB macro 63
 IOBSENS fields with macro instruction 69
 I/O device characteristics 100
 I/O supervisor
 appendages
 abnormal end (XCE) 55
 channel end (CE) 56
 end-of-extent 56
 entry points 52
 page fix (PGFX) 53
 program-controlled interrupt (PCI) 55
 register usage 51,52
 returns 52
 start I/O (SIO) 54
 use with EXCP 43

J

JFCB (job file control block)
 mapping macro (IEFJFCBN) 100
 processing 105
 (see also RDJFCB)
 JFCBMASK+4 field of JFCB 104
 job file control block
 (see JFCB)
 job queue 104

K

KEYLEN operand of DCB macro 65

L

LABEL= operand of DD statement, password protected data set 91,90
LNKX operand with **INDEX CAMLST** macros 15
LOCATE and **CAMLST** macro instructions 2-8
 with **BLOCK** operand 8
 reading a catalog entry by name (**NAME** operand)
 for data sets 3
 for generation data sets 5
 using alias name 6
 reading a catalog entry by **TTR** 8
LPALIB system generation macro 50

M

MACRF=(E), **DCB** operand for **EXCP** 67
 macro specifications for use with **EXCP** 60
 macros
ATLAS 67-72
CATALOG 16-23
CLOSE
 with **EXCP** 75
 with **XDAP** 83
CVT 100
DEBCHK 107
EOV
 with **EXCP** 74
 with **XDAP** 83
EXCP 67
IEFJFCBN 100
IEFUCBOB 99
INDEX 9-16
LOCATE 2-8
OPEN
 with **EXCP** 66
 with **XDAP** 80
PROTECT 92
RDJFCB 105
XDAP 81
 maintaining the **PASSWORD** data set 89
 (*see also* **PROTECT** macro)
 maintaining the system catalog 2-23
 maintaining the volume table of contents (**VTOC**) 33-41
 mapping macros
CVT 100
IEFJFCBN 100
IEFUCBOB 100
MODE operand of **DCB** macro 66
 modification of a channel program during execution 48
 multivolume direct and index-sequential data sets 67

N

nonpageable region/partition, **EXCP** operations in 45
NOPWREAD 93,90,91
NOWRITE 93,90

O

OBTAIN macro 34-36
 obtaining a sector number (**RPS** devices) 87
OPEN macro instruction
 with **EXCP** 66
 dummy data set restriction 66
 label processing 66
 procedures performed 66
 volume disposition 66
TYPE=J 104
 opening a **VTOC** to change its contents restriction 107
OPENJ (**OPEN**, **TYPE=J**) 104
OPTCD=Z 62
 output data sets, maintaining **DCBBLKCT** with **EXCP** 64

P

page fix (**PGFX**) appendage 53
 pageable region/partition
 (*see* translation of channel program)
 password
 control 93
 parameter list 94
 add a record 95
 delete a record 97
 list a record 97
 replace a record 96
 protection mode indicator 93
 record 90
 secondary 93
 standard label restriction 90
PASSWORD data set
 characteristics 90
 creating 90
 password protecting your data sets 89
 password protection processing
 counter maintenance 92
 data set concatenation 92
 termination 91
 volume switching 91
PCI (program controlled interrupt)
 appendage 55
 modify interface 55
 operand of **DCB** 62
 parameter list 55
PGFX (page fix)
 appendage 53
 operand of **DCB** 62
 posting of completion code in **ECB**
EXCP 78
XDAP 84
 printer image 115
 forms control buffer (**FCB**) 117
 universal character set (**UCS**) 115
 program controlled interrupt (**PCI**) appendage 55
PROTECT macro instruction 92
 parameter list 94-99
 return codes 98
 specification 94

- protection mode indicator 93
- PRTSP operand of DCB macro 66
- PURGE
 - chain 114
 - macro instruction 111
 - adding to macro library 110
 - definition 110
 - specification 111
- PWREAD 93,90
- PWWRITE 93,90



- RDJFCB macro instruction 105
 - coding examples 106
 - exceptional return codes 107
 - exit list entry for 106
 - specification 105
- read a job file control block (JFCB) 105
- reading a block from the catalog 2
- reading a catalog block
 - using alias name 6
 - coding example 7
 - exceptional returns 7
 - macro specification 6
 - using a data set name 2
 - coding example 3
 - exceptional returns 4
 - macro specification 3
 - using a generation data set name 4
 - coding example 5
 - exceptional return codes 4
 - macro specification 5
 - using a relative block address 7
 - coding example 8
 - exceptional return codes 4
 - macro specifications 8
- reading and modifying a JFCB 104
- READPSWD module 90
- recataloging a data set 21-23
 - coding example 22
 - exceptional return codes 18
 - macro specification 22
- RECFM operand of DCB macro 63
- recovering from permanent I/O errors 67
 - (see ATLAS macro instruction)
- register
 - conventions for appendages 52
 - usage by I/O supervisor with EXCP 51
- related channel programs 49
- relative generation number 4-5
- RENAME macro 38-43
- rename status code 40
- renaming a data set 38-41
 - coding example 39
 - exceptional return codes 40
 - macro specification 39
 - with password protection 41
 - status code 40
- request queue element, illustration of 52
 - (see also RQE)

- RESTORE macro instruction 110
 - adding to macro library 111
 - definition 111
 - specification 113
- restoring IOBs 114
- return codes
 - (see exceptional return codes)
- rotational position sensing (RPS) devices, with
 - XDAP 82,87
- RPS
 - (see rotational position sensing)
- RQEs
 - freeing 111
 - illustration of 52
 - restoring 113



- SCRATCH macro 36-38
- scratch status code 38
- scratching a data set 36-38
 - coding example 37
 - exceptional return codes 38
 - macro specification 37
 - with password protection 38
 - when volume not mounted 37
- secondary password 93
- sector
 - address in XDAP macro 82
 - conversion routine (IEC0SCR1) 87
- SIO (start I/O)
 - appendage 54
 - operand of DCB 62
- STACK operand of DCB macro 66
- standard label restriction, password
 - data sets 90
- stand-alone seek for 2314 and 2319 48
- start I/O appendages 54
- status code
 - deleting a multivolume data set 38
 - renaming a multivolume data set 40
- SVCLIB system generation macro instruction 50
- system catalog, maintaining 2-23
 - using CATALOG macro 16-23
 - using INDEX macro 9-16
 - using LOCATE macro 2-8
- system control blocks, macros for
 - mapping 99
 - CVT 100
 - IEFJFCBN 100
 - IEFUCBOB 99
 - modifying
 - DEBCHK, TYPE=ADD, DELETE, PURGE 107
 - OPEN, TYPE=J 105
 - PURGE 110
 - RDJFCB 105
 - RESTORE 110
 - obtain information from
 - DEBCHK, TYPE=VERIFY 107
 - DEVTYPE 100

system macro instructions 99
(*see also* system control blocks, macros for)
system work area data set (SWADS) 104
SYS1.
 SYSJOBQE 104
 LPALIB 50
 SVCLIB 50

T

TIC (transfers-in-channel) command, restriction 56
translation of channel programs by I/O supervisor
 extended 54
 in nonpageable regions 45
 normal 54
true name longer than alias 5
TRTCH operand of DCB macro 66
TTR 85

U

UCB (unit control block)
 getting information from
 (*see* DEVTYPE macro)
 mapping macro (IEFUCBOB) 99
UCS (universal character set) image 115
 for 1403 printer 116
 for 3211 printer 116
UEX (unit exception) 56
uncataloging a data set 20-21
 retaining index levels 20
 coding example 20
 exceptional return codes 18
 macro specification 20
 removing index levels 20-21
 coding example 21
 exceptional return codes 18
 macro specification 21
universal character set (UCS) image, adding to
 SYS1.IMAGELIB 115
unit check with ATLAS 69
unit control block (UCB) mapping macro (IEFUCBOB) 99

V

validating the DEB 107
virtual=real
 (*see* nonpageable region)
volume control block 2,23
 pointer entry 27
volume index control entry 23
volume table of contents (VTOC), maintaining
 using OBTAIN macro 34-36
 using RENAME macro 38-43
 using SCRATCH macro 36-38
volume list
 in catalog maintenance 2
 definition 2

 rename status code 40
 scratch status code 38
 volume status code 38,40
 volume switching 66
VS1 use of SVCLIB for I/O supervisor appendages 50
VS2 use of LPALIB for I/O supervisor appendages 50
VTOC
 (*see* volume table of contents)

W

WAIT macro instruction
 with EXCP 45
WLR (wrong-length record)
 (*see* channel end appendage)
"WRITE" protection mode indicator 38,41

X

XCE (abnormal end)
 appendage 57
 operand of DCB macro 62
XDAP macro instruction 79
 control blocks used with
 channel program 84
 ECB 83
 IOB 83
 macros required with
 CLOSE 83
 DCB 80
 EOV 83
 OPEN 80
 specification 81
XDAP channel program 85
XCE (abnormal end)
 appendage 57
 operand of DCB 62

1403 printer
 UCS image 115
 coding example 116
2314 stand-alone seek 48
2319 stand-alone seek 48
3211 printer
 FCB image 117
 coding example 118
 UCS image 115
 coding example 116

IBM CORPORATION
1100 N. WASHINGTON AVE
ARMONK, N.Y. 10504



International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

READER'S COMMENT FORM

OS/VS Data Management for
System Programmers

GC28-0631-I

Your comments about this publication will help us to produce better publications for your use. If you wish to comment, please use the space provided below, giving specific page and paragraph references.

Please do not use this form to ask technical questions about the system or equipment or to make requests for copies of publications. Instead, make such inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Reply requested

Yes

No

Name _____

Job Title _____

Address _____

_____ Zip _____

No postage necessary if mailed in the U S A

YOUR COMMENTS, PLEASE . . .

This publication is one of a series which serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

fold

fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.



POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. D78

fold

fold



International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

COPIED BY TELETYPE UNIT FOR SYSTEMS ADMINISTRATION

READER'S COMMENT FORM

OS/VS Data Management for
System Programmers

GC28-0631-I

Your comments about this publication will help us to produce better publications for your use. If you wish to comment, please use the space provided below, giving specific page and paragraph references.

Please do not use this form to ask technical questions about the system or equipment or to make requests for copies of publications. Instead, make such inquiries or requests to your IBM representative or to the IBM Branch Office serving your locality.

Reply requested

Yes

No

Name _____

Job Title _____

Address _____

_____ Zip _____

No postage necessary if mailed in the U S A

YOUR COMMENTS, PLEASE . . .

This publication is one of a series which serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

fold

fold

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.



POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. D78

fold

fold



International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)