OS/390

DFSMSrmm Application Programming Interface

SC26-7332-00

OS/390

DFSMSrmm Application Programming Interface

SC26-7332-00

Note!
Before using this information and the product it supports, be sure to read the general information under “Notices” on page 85.

First Edition, September 2000

This edition applies to Version 2 Release 10 of OS/390 (5647-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC26-7272-01.

Contents

Figures

Tables

About This Book

This book is intended for application programmers who use the DFSMSrmm™ application programming interface to obtain information about DFSMSrmm-managed resources.

Refer to:

- "Chapter 1. Using the DFSMSrmm Application Programming Interface" on page 1 for information on the EDGXCI macro you use for communication between your application program and DFSMSrmm.
- "Chapter 2. Programming Guidelines" on page 15 for guidelines for setting up communications.
- "Chapter 3. Receiving Output Data in the Output Buffer" on page 27 for information on the data that the DFSMSrmm application programming interface returns.

Required Product Knowledge

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

- The RMM TSO subcommand and operands
- Macros to communicate between programs

Referenced Publications

The following publications have additional information about DFSMSrmm:

Publication Title	Order Number
OS/390 DFSMSrmm Command Reference Summary	SX26-6020
OS/390 DFSMSrmm Diagnosis Guide	SY27-7612
OS/390 DFSMSrmm Guide and Reference	SC26-7333
OS/390 DFSMSrmm Implementation and Customization Guide	SC26-7334
OS/390 DFSMSrmm Reporting	SC26-7335

This book also refers to the following publications:

Title	Order Number
OS/390 Planning for Installation	GC28-1726
OS/390 MVS System Messages, Volume 2 (ASB-EZM)	GC28-1785

Notational Conventions

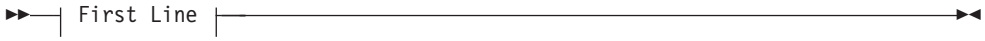
This section explains the notational conventions used in this book.

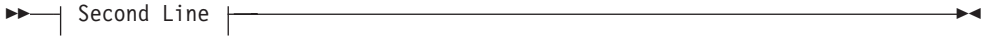
How to Read Syntax Diagrams

Throughout this library, diagrams are used to illustrate the programming syntax. Keyword parameters are parameters that follow the positional parameters. Unless otherwise stated, keyword parameters can be coded in any order. The following list tells you how to interpret the syntax diagrams:

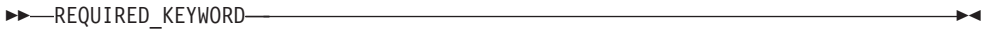
- Read the diagrams from left-to-right, top-to-bottom, following the main path line. Each diagram begins on the left with double arrowheads and ends on the right with two arrowheads facing each other.

- If a diagram is longer than one line, each line to be continued ends with a single arrowhead and the next line begins with a single arrowhead.





- Required keywords and values appear on the main path line. You must code required keywords and values.



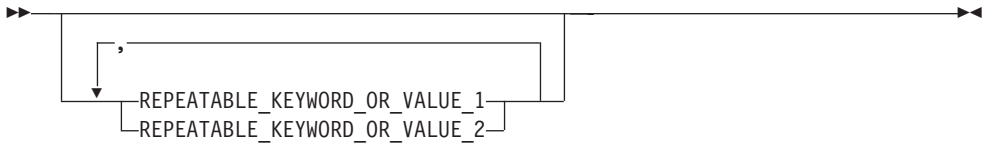
  If several mutually exclusive required keywords or values exist, they are stacked vertically in alphanumeric order.

- Optional keywords and values appear below the main path line. You can choose not to code optional keywords and values.

  If several mutually exclusive optional keywords or values exist, they are stacked vertically in alphanumeric order below the main path line.

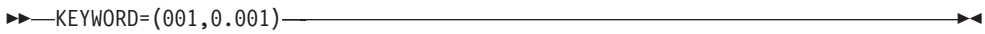
- An arrow returning to the left above a keyword or value on the main path line means that the keyword or value can be repeated. The comma means that each keyword or value must be separated from the next by a comma.

- An arrow returning to the left above a group of keywords or values means more than one can be selected, or a single one can be repeated.



- A word in all uppercase is a keyword or value you must spell exactly as shown. In this example, you must code ***KEYWORD***.

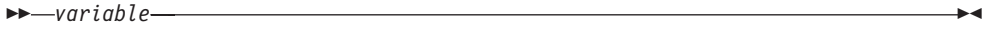
  If a keyword or value can be abbreviated, the abbreviation is discussed in the text associated with the syntax diagram.
- If a diagram shows a character that is not alphanumeric (such as parentheses, periods, commas, and equal signs), you must code the character as part of the syntax. In this example, you must code ***KEYWORD=(001,0.001)***.



- If a diagram shows a blank space, you must code the blank space as part of the syntax. In this example, you must code ***KEYWORD=(001 FIXED)***.

- Default keywords and values appear above the main path line. If you omit the keyword or value entirely, the default is used.

- A word in all lowercase italics is a *variable*. Where you see a variable in the syntax, you must replace it with one of its allowable names or values, as defined in the text.



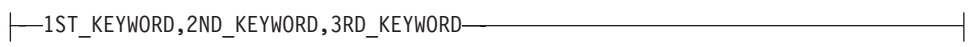
- References to syntax notes appear as numbers enclosed in parentheses above the line. Do not code the parentheses or the number.

Notes:

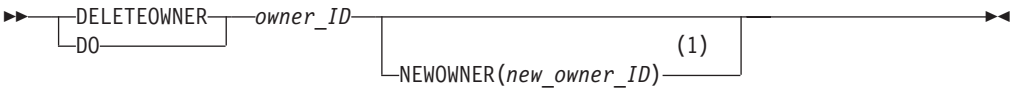
1 An example of a syntax note.

- Some diagrams contain *syntax fragments*, which serve to break up diagrams that are too long, too complex, or too repetitious. Syntax fragment names are in mixed case and are shown in the diagram and in the heading of the fragment. The fragment is placed below the main diagram.

Syntax Fragment:



The following figure shows an example of a syntax diagram.



Notes:

1 Must be specified if the owner owns one or more volumes.

Figure 1. Example of the DFSMSrmm DELETEVOLUME Syntax Diagram

The possible valid versions of the RMM DELETEOWNER command are:

```
RMM DELETEOWNER owner
RMM DO          owner
RMM DELETEOWNER owner NEWOWNER(new_owner)
RMM DO          owner NEWOWNER(new_owner)
```

How to Abbreviate Commands and Operands

The TSO abbreviation convention applies for all DFSMSrmm commands and operands. The TSO abbreviation convention requires you to specify as much of the command name or operand as is necessary to distinguish it from the other command names or operands.

Some DFSMSrmm keyword operands allow unique abbreviations. All unique abbreviations are shown in the command syntax diagrams.

How to Use Continuation Characters

The symbol **-** is used as the continuation character in this book. You can use either **-** or **+**.

- Do not ignore leading blanks on the continuation statement

+ Ignore leading blanks on the continuation statement

Delimiters

When you type a command, you must separate the command name from the first operand by one or more blanks. You must separate operands by one or more blanks or a comma. Do not use a semicolon as a delimiter because any character you enter after a semicolon is ignored.

Character Sets

To code job control statements, use characters from the character sets in Table 1. Table 2 lists the special characters that have syntactical functions in job control statements.

Table 1. Character Sets

Character Set	Contents	
Alphanumeric	Alphabetic Numeric	Capital A through Z 0 through 9
National (See note)	“At” sign Dollar sign Pound sign	@ (Characters that can be $ represented by hexadecimal # values X'7C', X'5B', and X'7B')
Special	Comma Period Slash Apostrophe Left parenthesis Right parenthesis Asterisk Ampersand Plus sign Hyphen Equal sign Blank	, . / ' ( ) * & + - =
EBCDIC text	EBCDIC printable character set	Characters that can be represented by hexadecimal X'40' through X'FE'
Note: The system recognizes the following hexadecimal representations of the U.S. National characters; @ as X'7C'; $ as X'5B'; and # as X'7B'. In countries other than the U.S., the U.S. National characters represented on terminal keyboards might generate a different hexadecimal representation and cause an error. For example, in some countries the $ character may generate a X'4A'.		

Table 2. Special Characters Used in Syntax

Character	Syntactical Function
,	To separate parameters and subparameters
=	To separate a keyword from its value, for example, BURST=YES
( )	To enclose subparameter list or the member name of a PDS or PDSE
&	To identify a symbolic parameter, for example, &LIB
&&	To identify a temporary data set name, for example, &&TEMPDS, and, to identify an in-stream or sysout data set name, for example, &&PAYOUT
.	To separate parts of a qualified data set name, for example, A.B.C., or parts of certain parameters or subparameters, for example, nodename.userid
*	To refer to an earlier statement, for example, OUTPUT=*.name, or, in certain statements, to indicate special functions: //label CNTL * //ddname DD * RESTART=* on the JOB statement

Table 2. Special Characters Used in Syntax (continued)

Character	Syntactical Function
'	To enclose specified parameter values which contain special characters
(blank)	To delimit fields

Accessing OS/390 DFSMS Books on the Internet

In addition to making softcopy books available on CD-ROM, IBM provides access to unlicensed OS/390 softcopy books on the Internet. To find OS/390 books on the Internet, first go to the OS/390 home page: http://www1.s390.ibm.com/os390/

From this Web site, you can link directly to the OS/390 softcopy books by selecting the Library icon. You can also link to IBM Direct to order hardcopy books.

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other DFSMS documentation:

- Send your comments by e-mail to:
  - IBMLink from US: starpubs@us.ibm.com
  - IBMLink from Canada: STARPUBS at TORIBM
  - IBM Mail Exchange: USIB3VVD at IBMMAIL
  - Internet: starpubs@us.ibm.com

  Be sure to include the name of the book, the part number of the book, version and product name, and if applicable, the specific location of the text you are commenting on (for example, a page number or a table number).
- Fill out one of the forms at the back of this book and return it by mail or by giving it to an IBM representative. If the form has been removed, address your comments to IBM Corporation, RCF Processing Department M86/050, 5600 Cottle Road, San Jose, California 95193-0001, U.S.A.

Summary of Changes

The summary of changes informs you of changes to this book. Revision bars (|) in the left margin of the book indicate changes from the previous edition.

Summary of Changes for SC26-7332-00 OS/390 DFSMSrmm Application Programming Interface

This book contains information previously presented in *DFSMSrmm Application Programming Interface*, SC26-7272-01.

The following sections summarize the changes to that information.

New Information

This edition includes the following new information:

- Added new structured field introducers.

Changed Information

The following information was changed in this edition:

- Updated output data in “Chapter 3. Receiving Output Data in the Output Buffer” on page 27 to include new structured field introducers.
- Updated information in “SFIs for Subcommand Output Data” on page 58 by consolidating information into a single table.

Summary of Changes for SC26-7272-01 DFSMS/MVS Version 1 Release 5 DFSMSrmm Application Programming Interface

This book contains information previously presented in *DFSMS/MVS® Version 1 Release 5 DFSMSrmm Application Programming Interface*, SC26-7272-00.

The following sections summarize the changes to that information.

New Information

This edition includes the following new information:

- Added information about new structured field introducers.

Chapter 1. Using the DFSMSrmm Application Programming Interface

DFSMSrmm is an OS/390® feature. Use the DFSMSrmm application programming interface (API) to read, extract, and update data in the DFSMSrmm control data set. You can use the data to create reports or implement automation.

Use macro EDGXCI as described in “EDGXCI: Call DFSMSrmm Interface” on page 3 to define a parameter list to call the DFSMSrmm application programming interface. Use macro EDGXCI to pass any supported RMM TSO subcommand to DFSMSrmm. See “Supported RMM TSO Subcommands” on page 2 for a list of supported RMM TSO subcommands. Figure 3 on page 12 is an example you can modify to communicate with the DFSMSrmm application programming interface.

Use macro EDGXSF as described in “EDGXSF: Structured Field Definitions” on page 76 to help you process the data that the DFSMSrmm application programming interface returns. The DFSMSrmm application programming interface returns data as structured fields in an output buffer that you define. Structured fields consist of these parts.

- A structured field introducer (SFI) that introduces the type of data, length, and characteristics of the data that the API returns,
- Data.

You can request that the API returns data in line format or field format as described in “SFI Data Format” on page 28. You can also request standard output or expanded output as described in “Types of Output” on page 31.

To use the DFSMSrmm application programming interface, you must have High Level Assembler installed on your system. *OS/390 Planning for Installation* provides information about the level of High Level Assembler required for DFSMS.

Supported RMM TSO Subcommands

The DFSMSrmm API supports all the RMM TSO subcommands as shown in Table 3.

Table 3. RMM TSO Subcommands

Group	Subcommand	Abbrev	Function
Add	ADDBIN	AB	Add bin number information
	ADDDATASET	AD	Add data set information
	ADDOWNER	AO	Add owner information
	ADDPRODUCT	AP	Add software product information
	ADDRACK	AR	Add shelf location information
	ADDVOLUME	AV	Add volume information
	ADDVRS	AS	Add a vital record specification
Change	CHANGEDATASET	CD	Change data set information
	CHANGEOWNER	CO	Change owner information
	CHANGEPRODUCT	CP	Change software product information
	CHANGEVOLUME	CV	Change volume information
Delete	DELETEBIN	DB	Delete bin number information
	DELETEDATASET	DD	Delete data set information
	DELETEOWNER	DO	Delete owner information
	DELETEPRODUCT	DP	Delete software product information
	DELETERACK	DR	Delete shelf location information
	DELETEVOLUME	DV	Release a volume and delete volume
	DELETEVRS	DS	Delete a vital record specification information
Get	GETVOLUME	GV	Request or assign a volume
List	LISTBIN	LB	Display bin number information
	LISTCONTROL	LC	Display parmlib options and control information
	LISTDATASET	LD	Display data set information
	LISTOWNER	LO	Display owner information
	LISTPRODUCT	LP	Display software product information
	LISTRACK	LR	Display shelf location information
	LISTVOLUME	LV	Display volume information
	LISTVRS	LS	Display vital record specification information
Search	SEARCHBIN	SB	Create a list of bin numbers
	SEARCHDATASET	SD	Create a list of data sets
	SEARCHPRODUCT	SP	Create a list of software products
	SEARCHRACK	SR	Create a list of rack numbers
	SEARCHVOLUME	SV	Create a list of volumes
	SEARCHVRS	SS	Create a list of vital record specifications

Refer to *OS/390 DFSMSrmm Guide and Reference* for details on these subcommands.

Note: When you use the DFSMSrmm application programming interface, you must specify the subcommand as a single, continuous string of characters rather than as multiple input lines.

Using the EDGXCI Macro

Follow these steps to obtain information from DFSMSrmm using the EDGXCI macro.

1. Use EDGXCI MF=(L,addr) to save space in your dynamic area for the parameter list.
2. Save the address of an output buffer that the application programming interface uses.
3. Load the DFSMSrmm API module, EDGXAPI, and then save the address of the module.
4. Create the subcommand that you want to process.
5. Use the EDGXCI macro to complete the parameter list and call the DFSMSrmm application programming interface.
6. Use EDGXCI with OPERATION=CONTINUE as needed to get more data for the current subcommand.
7. Use EDGXCI with OPERATION=RELEASE to free resources that are obtained by the DFSMSrmm API module.
8. Delete the EDGXAPI module that you loaded.

EDGXCI: Call DFSMSrmm Interface

Use the EDGXCI macro in your application program (the caller) to:

- Define a parameter list.
- Set parameters in the list.
- Change parameters in the list.
- Call the DFSMSrmm application programming interface module, EDGXAPI.

EDGXCI Macro — Call to DFSMSrmm API

Application Program call to DFSMSrmm API.

EDGXCI Environment

The requirements for the caller are:

Minimum authorization:	Non-APF authorized, problem state and key (0-8).
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
Control parameters:	Control parameters must be in the primary address space.

EDGXCI Programming Requirements

The caller must load the DFSMSrmm API module, EDGXAPI, prior to using the execute or standard form of EDGXCI. The caller must delete EDGXAPI when the DFSMSrmm API is no longer needed.

The caller should also use the EDGXSF macro to define the structured fields that are used in the output.

EDGXCI Restrictions

The caller must not have functional recovery routines (FRRs) established.

EDGXCI Input Register Information

Before issuing the EDGXCI macro, ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
13	The address of a 72-byte standard save area in the primary address space

Before issuing the EDGXCI macro, no information is needed in any access register (AR) unless the access register is used in register notation for a particular parameter or as a base register.

EDGXCI Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

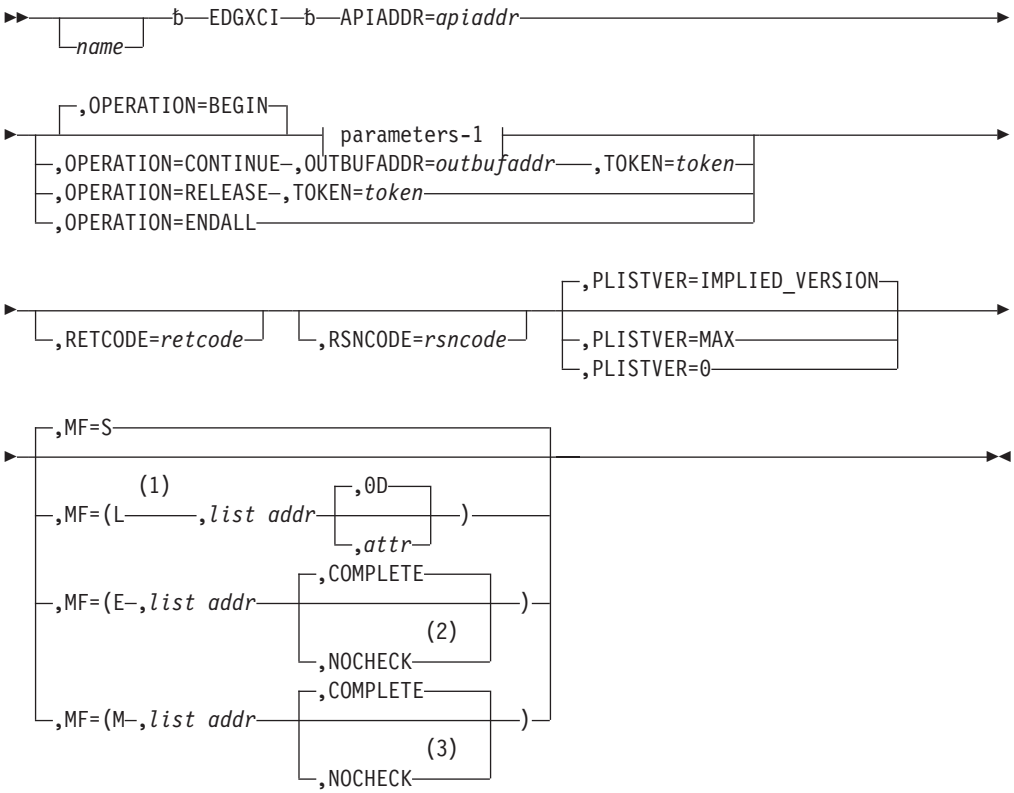
Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Some callers depend on register contents that remain the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

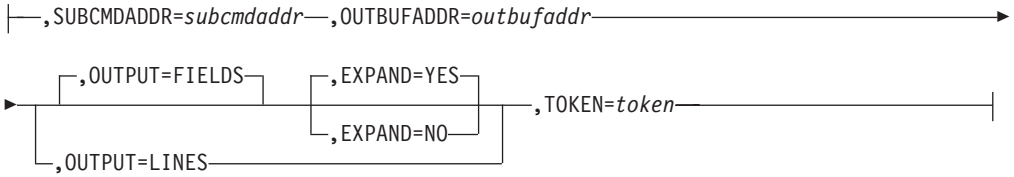
EDGXCI Syntax

Figure 2 on page 5 shows the syntax for the EDGXCI macro. You can use this macro to communicate with the DFSMSrmm application programming interface.

EDGXCI Macro



parameters-1:



Notes:

1 Only the PLISTVER parameter can be coded with MF=L.

2 When NOCHECK is specified with MF=E, all parameters are optional and the system does not supply defaults for omitted optional parameters.

3 When NOCHECK is specified with MF=M, all parameters are optional and the system does not supply defaults for omitted optional parameters.

Figure 2. EDGXCI Macro Syntax Diagram

EDGXCI Parameters

You can specify these parameters:

name
An optional symbol that starts in column 1. This is the name on the EDGXCI macro call. The name must conform to the rules for an ordinary assembler language symbol.

APIADDR=*apiaddr*

A required input parameter that contains the address of the DFSMSrmm API load module. The calling program is responsible for loading the DFSMSrmm API load module, saving, and then using the returned load address. Use the MVS™ LOAD service to obtain the DFSMSrmm API address.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,EXPAND=YES

,EXPAND=NO

When OUTPUT=FIELDS and OPERATION=BEGIN are specified, EXPAND is an optional parameter that specifies whether to expand the number of returned data fields to be the same as for the corresponding list type of subcommand. The default is EXPAND=YES.

,EXPAND=YES

Specify to expand the number of data fields to be the same as the corresponding list type of subcommand.

,EXPAND=NO

Specify to not expand the number of data fields for the subcommand.

,MF=S

,MF=(L,*list addr***)**

,MF=(L,*list addr,attr***)**

,MF=(L,*list addr***,0D)**

,MF=(E,*list addr***)**

,MF=(E,*list addr***,COMPLETE)**

,MF=(E,*list addr***,NOCHECK)**

,MF=(M,*list addr***)**

,MF=(M,*list addr***,COMPLETE)**

,MF=(M,*list addr***,NOCHECK)**

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro. This builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the macro list form with the macro execute form for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

Use MF=M together with the list form and execute form of the macro for service routines that need to provide different options according to user-provided input.

Use the list form to define a storage area. Use the modify form to set the appropriate options. Then use the execute form to call the service.

IBM® recommends that you use the modify and execute forms of EDGXCI in the following order:

1. Use EDGXCI ...MF=(M,list-addr,COMPLETE) and specify all the required parameters and any appropriate optional parameters.
2. Use EDGXCI ...MF=(M,list-addr,NOCHECK) and specify the parameters that you want to change.
3. Use EDGXCI ...MF=(E,list-addr,NOCHECK) to execute the macro.

,*list addr*
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (1)-(12).

,*attr*
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of X'0F' to force the parameter list to a word boundary or X'0D' to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of X'0D'.

,COMPLETE
Specifies that the system should check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

,OPERATION=BEGIN

,OPERATION=CONTINUE

,OPERATION=RELEASE

,OPERATION=ENDALL
An optional parameter that describes the processing of the current subcommand. The default is OPERATION=BEGIN.

,OPERATION=BEGIN
Specify BEGIN to start a new subcommand.

,OPERATION=CONTINUE
Specify CONTINUE to continue the current subcommand.

,OPERATION=RELEASE
Specify when you want the token and all its associated resources to be released.

,OPERATION=ENDALL
Specify when you want to end all operations by releasing all tokens and all resources.

,OUTBUFADDR=*outbufaddr*
When OPERATION=BEGIN is specified, OUTBUFADDR=*outbufaddr* is a required input parameter that contains the address of your output buffer, which is used for both data and messages. It must be at least 4096 bytes in length. The first four bytes of the buffer must contain the length of the buffer, including the four bytes of the length.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,OUTBUFADDR=*outbufaddr*
When OPERATION=CONTINUE is specified, OUTBUFADDR=*outbufaddr* is a required input parameter that contains the address of your output buffer, which is used for both data and messages. It must be at least 4096 bytes in length. The first four bytes of the buffer must contain the length of the buffer, including the four bytes of the length.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,OUTPUT=FIELDS

,OUTPUT=LINES
When OPERATION=BEGIN is specified, OUTPUT is an optional parameter that specifies the format of the returned data. The default is OUTPUT=FIELDS.

,OUTPUT=FIELDS
Specify when you want data returned in field format.

,OUTPUT=LINES
Specify when you want data returned in line format. Search output is always returned in standard form when OUTPUT=LINES is specified.

,PLISTVER=IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. Specify PLISTVER on all macro forms used for a request and with the same value on all of the macro forms. The PLISTVER values are:

- **IMPLIED_VERSION** which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX** which allows you to change to the largest size currently possible. This size might grow from release to release and affect the amount of storage that your application program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is large enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,SUBCMDADDR=*subcmdaddr*
When OPERATION=BEGIN is specified, SUBCMDADDR=*subcmdaddr* is a required input parameter that contains the address of the input subcommand. The subcommand consists of a halfword field followed by the subcommand text. The halfword field must contain the length of the subcommand, including both the halfword field and the subcommand text. The maximum value is 32 761.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,TOKEN=*token*
When OPERATION=BEGIN is specified, TOKEN=*token* is a required input parameter of a 4-byte area. The DFSMSrmm API creates a token and obtains resources for it, or the DFSMSrmm API reuses the token and the resources.

TOKEN is required even when MF=(E,label,NOCHECK) is specified, unless OPERATION=ENDALL is also specified.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,TOKEN=*token*
When OPERATION=CONTINUE is specified, TOKEN=*token* is a required input parameter of a 4-byte area containing the token used to begin the subcommand. The DFSMSrmm API uses the resources for the token to continue the subcommand.

TOKEN is required even when MF=(E,label,NOCHECK) is specified, unless OPERATION=ENDALL is also specified.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,TOKEN=*token*
When OPERATION=RELEASE is specified, TOKEN=*token* is a required input parameter of a 4-byte area containing a token. The DFSMSrmm API releases the resources for the token, releases the token, and clears the 4-byte area.

TOKEN is required even when MF=(E,label,NOCHECK) is specified, unless OPERATION=ENDALL is also specified.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

EDGXCI Return and Reason Codes

When the EDGXCI macro returns control to your application program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The EDGXCI macro returns the following types of return codes and reason codes:

- Return and reason codes that are associated with the processing of your subcommand. These return and reason codes are the same ones that DFSMSrmm returns when you issue a subcommand request. Refer to *OS/390 DFSMSrmm Guide and Reference* for more information about these return and reason codes.
- Return codes and reason codes that are issued by the API. The API returns:
  - Return code 0 and reason code 0 when processing has completed successfully.
  - Return code 0 and reason code 4 when the output buffer is full and more information is available.
  - Any return code higher than 100 when an error has occurred.

Table 4 identifies the decimal return and reason codes.

Table 4. Return and Reason Codes for the EDGXCI Macro

Return Code	Reason Code	Meaning and Action
0	—	**Meaning**: Success. **Action**: Refer to the action provided with the specific reason code.
0	0	**Meaning**: EDGXCI command is successfully completed. **Action**: None required.
0	4	**Meaning**: There is more output waiting to be given to you. **Action**: After you have processed the output in your output buffer, use OPERATION=CONTINUE to get more output.
104	—	**Meaning**: Program error. An exception condition has been encountered, but the operation you requested was completed. The output results might not be acceptable to you. **Action**: Refer to the action provided with the specific reason code.
104	02	**Meaning**: There is nothing to CONTINUE. **Action**: None required.
108	—	**Meaning**: Program error. An error condition has been encountered, and the operation you requested was not successfully completed. **Action**: Refer to the action provided with the specific reason code.
108	02	**Meaning**: Required token is missing. **Action**: You need to use TOKEN=**token**
108	04	**Meaning**: Required address of the input subcommand is missing. **Action**: You need to use SUBCMDADDR=**subcmdaddr**
108	06	**Meaning**: Required address of your output buffer is missing. **Action**: Use OUTBUFADDR=**outbufaddr** to specify the parameter.

Table 4. Return and Reason Codes for the EDGXCI Macro (continued)

Return Code	Reason Code	Meaning and Action
108	08	**Meaning**: Your output buffer is less than 4096 bytes in size. **Action**: Obtain storage and set its length.
108	10	**Meaning**: Your output buffer is too small. The second word in your buffer contains the size you need. **Action**: Obtain the correct amount of storage and set its length.
108	12	**Meaning**: OPERATION parameter is invalid. **Action**: Use OPERATION= to specify the parameter; check your program for incorrect modifying of the parameter list.
108	14	**Meaning**: OUTPUT parameter is invalid. **Action**: Use OUTPUT= to specify the parameter; check your program for incorrect modifying of the parameter list.
108	16	**Meaning**: EXPAND parameter is invalid. **Action**: Use EXPAND= to specify the parameter; check your program for incorrect modifying of the parameter list.
108	56	**Meaning**: The token is already in use. **Action**: Use TOKEN=**token** to specify a token that is not in use.
108	58	**Meaning**: OUTPUT=FIELDS is not supported for the subcommand specified by SUBCMDADDR=**subcmdaddr**. **Action**: Use OUTPUT=LINES or specify a different subcommand.
108	60	**Meaning**: The length of the subcommand specified by SUBCMDADDR=**subcmdaddr** is too large. **Action**: Use a smaller subcommand.
112	—	**Meaning**: Environmental error. A limit, such as a storage limit, was exceeded. The operation you requested was not successfully completed. **Action**: Refer to the action provided with the specific reason code.
112	02	**Meaning**: Unable to obtain sufficient work area storage. **Action**: Remove the cause of the short-on-storage condition or request a larger region size. Rerun your program.
116	—	**Meaning**: System error. An error caused by the system, rather than your program, has been encountered. The operation you requested was not successfully completed. **Action**: Refer to the action provided with the specific reason code.
116	02	**Meaning**: DFSMSrmm is not installed. **Action**: Ensure DFSMSrmm is installed and active before running your program.

Table 4. Return and Reason Codes for the EDGXCI Macro (continued)

Return Code	Reason Code	Meaning and Action
116	04	**Meaning**: A call to a system service has resulted in a non-zero return code. DFSMSrmm has placed the return code and the associated reason code as structured fields in your output buffer. **Action**: Retry the subcommand after the cause of the error has been corrected or removed.
116	06	**Meaning**: An abnormal end has occurred. **Action**: Remove the cause of the abnormal end. Rerun your program.

EDGXCI Example

You can modify the example shown in Figure 3 to:

- Obtain space for your output buffer in your work area in dynamic storage.
- Obtain space for the parameter list in your work area in dynamic storage.
- Specify subcommands that have the following format:
  - The subcommand is prefixed by a two-byte length.
  - The subcommand is specified as a single input string.
- Use addresses that are pointer fields.
- Reuse the same parameter list for many requests.
- Reuse your 4-byte token area by specifying TOKEN= on all EXECUTE forms of EDGXCI. Your 4-byte token area is updated on return from the DFSMSrmm API.
- Make the list form parameter list large enough for all the parameters you might specify by using PLISTVER=MAX on the execute form of the EDGXCI macro.

```
YOURPGM  CSECT
R0       EQU   0
R1       EQU   1
R3       EQU   3
R4       EQU   4
R9       EQU   9
R11      EQU   11
R12      EQU   12
R13      EQU   13
R15      EQU   15
*        ..
         USING *,R11
         USING WORKDS,R12
         LA    R13,REGSAVE        Point to register save area
*        ..
*        ..
         LA    R0,OUTBUFWK        Save the
         ST    R0,APIOUTB@           address of output buffer
```

Figure 3. (Part 1 of 3) Communicating with the DFSMSrmm API

```
*************************************************************
*        Load the API module                                **
*************************************************************
         LOAD  EP=EDGXAPI
         ST    R0,APIMOD@         Save API module address
*        ..
         XC    MYTOKEN,MYTOKEN    Ensure no token yet
         LA    R4,LISTV@          List volume subcmd address
         BAL   R9,BEGINCMD        Begin the command
*        ..
*************************************************************
*        Going to reuse the resources, instead of releasing**
*        resources obtained by the API for the 1st BEGIN   **
*************************************************************
         LA    R4,SEARCHD@        Search subcmd address
         BAL   R9,BEGINCMD        Begin the command
*        ..
         BAL   R9,MOREDATA        Get more data for search
*        ..
         BAL   R9,RELEASE         All done, release resources
*        ..
*************************************************************
*        Delete the API module                              **
*************************************************************
         DELETE EP=EDGXAPI
*        ..
*************************************************************
**       Call API to begin a new subcommand                 **
*************************************************************
BEGINCMD DS    0H
CALL1    EDGXCI  MF=(E,MYPL),PLISTVER=MAX,                   X
               APIADDR=APIMOD@,OPERATION=BEGIN,              X
               TOKEN=MYTOKEN,                                X
#              SUBCMDADDR=(R4),OUTBUFADDR=APIOUTB@
         BR    R9                 Return
*************************************************************
**       Call API to get more data for current subcommand  **
*************************************************************
MOREDATA DS    0H
CALL2    EDGXCI  MF=(E,MYPL,NOCHECK),PLISTVER=MAX,           X
               OPERATION=CONTINUE,TOKEN=MYTOKEN
         BR    R9                 Return
```

Figure 4. (Part 2 of 3) Communicating with the DFSMSrmm API

```
*************************************************************
**       Call API to release resource such as storage and  **
**            loaded modules.                              **
*************************************************************
RELEASE  DS    0H
REL1     EDGXCI  MF=(E,MYPL,NOCHECK),PLISTVER=MAX,           X
               OPERATION=RELEASE,TOKEN=MYTOKEN
         BR    R9                 Return
*************************************************************
**       SEARCH DATA SET SUBCOMMAND                        **
*************************************************************
SEARCHD  DS    0C
         DC    AL2(SEARCHDL)
         DC    C'SEARCHDATASET ....'
SEARCHDL EQU   *-SEARCHD
SEARCHD@ DC    A(SEARCHD)
*************************************************************
**       LISTVOLUME SUBCOMMAND                             **
*************************************************************
LISTV    DS    0C                 Listv command buffer
         DC    AL2(LISTVL)        Length of command
         DC    C'LISTVOLUME ....'
LISTVL   EQU   *-LISTV            Length of command
LISTV@   DC    A(LISTV)           Address of command
*        ..
*************************************************************
**       PROGRAM WORK AREA                                 **
*************************************************************
WORKDS   DSECT
APIOUTB@ DS    A                  Pointer to output buffer
APIMOD@  DS    A                  Address of the API module
REGSAVE  DS    18F                Save area
MYTOKEN  DS    CL4                Token from the API
*************************************************************
**       PARAMETER LIST DEFINITION                         **
*************************************************************
         EDGXCI MF=(L,MYPL,0D),PLISTVER=MAX PLIST area
         DS    0D
OUTBUFWK DS    CL4096             Output buffer area
*************************************************************
**       STRUCTURED FIELD DEFINITIONS                      **
*************************************************************
SFDEFDS  DSECT
         EDGXSF
         END
```

Figure 5. (Part 3 of 3) Communicating with the DFSMSrmm API

Note: Macro continuation characters need to be in column 72.

Chapter 2. Programming Guidelines

When you use the DFSMSrmm API, your application program should:

- Allocate a sufficient number of token areas.
- Allocate a sufficient number of output buffers.
- Allocate a sufficient number of parameter lists.
- Consider using a different output buffer for each subcommand request. Reuse an output buffer to begin a new subcommand request only when there is nothing in the buffer that you need.
- Use the correct token when continuing a subcommand or when releasing a particular set of resources.
- Reuse the resources associated with the token, especially when you are processing hundreds or thousands of subcommands.
- Reuse a token to begin a new subcommand only when you no longer need the information obtained from the previous request.
- Use the EDGXCI macro parameter OPERATION=RELEASE to release unneeded resources associated with a token.
- Use the EDGXCI macro parameter OPERATION=ENDALL to ensure that all resources are released. You must also set all the tokens associated with the resources to zeros to prevent using a token that does not contain correct information.

Specifying the Subcommand Input in EDGXCI

To obtain information from the DFSMSrmm control data set, specify a DFSMSrmm TSO subcommand as a single input line without the RMM command, as shown in Figure 6. Do not specify it as an RMM command with multiple input lines, as shown in Figure 7.

```
AV MLV001 STATUS(MASTER) EXPDT(98001) OWNER(IBMUSER) OWNERACCESS(UPDATE) RACK(ML0001)
```

Figure 6. Example of Specifying the DFSMSrmm API Subcommand

```
RMM  AV MLV001 STATUS(MASTER) EXPDT(98001) OWNER(IBMUSER)-
        OWNERACCESS(UPDATE) RACK(ML0001)
```

Figure 7. Example of Specifying the RMM TSO Subcommand

In addition, specify subcommands using fully specified subcommand operands and their values. Avoid abbreviating the subcommands or operands because they can change when new subcommand operands and values are added for new functions.

Using the CONTINUE Operation in EDGXCI

The DFSMSrmm API can return control back to your application program before returning all the data you expect because:

- There is no more room in the output buffer for the additional data.
- The API stops after returning data for a single resource when you issue a request using a SEARCH command and the OUTPUT=FIELDS parameter.
- There is no more data to return to your application program.

Write your application program to check the return codes and reason codes that the DFSMSrmm API returns to your application program. To receive more data, you need to use EDGXCI OPERATION=CONTINUE parameter.

When you issue requests specifying the LISTCONTROL subcommand or the SEARCH type subcommands, the DFSMSrmm API issues return code 0 and reason code 4.

- For requests specifying SEARCH type subcommands with the OUTPUT=LINES or LISTCONTROL subcommands for both LINES and FIELDS, the output buffer is too full to hold any more output data.

  You might consider increasing the size of your output buffer for SEARCH type of subcommands or LISTCONTROL subcommands.
- For SEARCH type subcommands with OUTPUT=FIELDS, the DFSMSrmm API stops after all of the data for a single resource, such as a data set or volume, has been placed in your output buffer, even though there might be room for another resource in your output buffer.

The DFSMSrmm API issues return code 4 and reason code 2 in response to a SEARCH type subcommand. The DFSMSrmm API issues these codes when the search limit you set for a DFSMSrmm subcommand has been reached. The API has placed all of the data it received from DFSMSrmm into your output buffer, but there might be more records to search. See “Limiting the Search for a Request” on page 50 for more information.

DFSMSrmm issues these return codes and reason codes when you have coded OPERATION=CONTINUE and there are no more records to search or because the search limit has been reached.

- Return code 0 and reason code 0
- Return code 4 and reason code 4
- Return code 4 and reason code 8

When you use OPERATION=CONTINUE, you might not receive more output data or you might receive only messages in your output buffer.

See “Controlling Output from List and Search Type Requests” on page 50 for an example of the interaction between the size of an output buffer, the amount of output data the API returns, and the LIMIT value you set.

Using Parameter Lists

Your application can use single or multiple parameter lists For example, your application program can use one parameter list for a SEARCH type of subcommand and another parameter list for a CHANGE type of subcommand.

You need to decide if your application is going to:

- Process subcommands serially or concurrently.
- Use single or multiple parameter lists for each subcommand.
- Reuse resources (tokens).

The following is a list of major variations on using parameter lists and tokens in your application program:

- Using a single parameter list and a single token area. See “Single Parameter List, Single Token Area”.
- Using a single parameter list and multiple token areas. See “Single Parameter List, Multiple Token Areas” on page 18.
- Using multiple parameter lists and a single token area. See “Multiple Parameter List, Single Token Area” on page 21.
- Using multiple parameter lists and multiple token areas. See “Multiple Parameter List, Multiple Token Areas” on page 22.

You can combine any of these variations to meet your application requirements. The examples in the sections that follow use inline code segments with shortened code lines for illustrative purposes.

Single Parameter List, Single Token Area

When you use a single parameter list and a single token:

- Only one subcommand request can be active at a time.
- An active subcommand request must be completed before beginning another subcommand request.

In Figure 8, your application program uses a single parameter list and a single token area. The example includes a BEGIN, CONTINUE, and RELEASE for each subcommand request because you are not reusing resources. The token for the second subcommand request is different from the first subcommand request because you are not reusing any resources and need a separate token for each request.

```
****************************************
** Start the first subcommand
****************************************
  XC    TOKENA,TOKENA            No resources/token yet
  LA    R4,SUBCMD1               Point to 1st subcommand
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=BEGIN,                X
        TOKEN=TOKENA,                                   X
        SUBCMDADDR=(R4),OUTBUFADDR=(R3)
  ...
```

Figure 8. (Part 1 of 2) Single Parameter List, Single Token Area

```
*****************************************
** Continue the subcommand
*****************************************
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=CONTINUE,             X
        TOKEN=TOKENA,                                   X
        OUTBUFADDR=(R3)
  ...
*****************************************
** Done with the subcommand, release
*****************************************
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=RELEASE,              X
        TOKEN=TOKENA
  ...
*****************************************
** Start the second subcommand
*****************************************
  LA    R4,SUBCMD2              Point to 2nd subcommand
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=BEGIN,                X
        TOKEN=TOKENA,                                   X
        SUBCMDADDR=(R4),OUTBUFADDR=(R3)
  ...
*****************************************
** Continue the subcommand
*****************************************
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=CONTINUE,             X
        TOKEN=TOKENA,                                   X
        OUTBUFADDR=(R3)
  ...
*****************************************
** Done with the subcommand, release
*****************************************
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=RELEASE,              X
        TOKEN=TOKENA
```

Figure 9. (Part 2 of 2) Single Parameter List, Single Token Area

The example includes the OPERATION=RELEASE parameter. When you use OPERATION=RELEASE, work areas that are used to contain data and pointers for the subcommand are released and you must obtain resources for the next subcommand request. You might improve performance by deleting the OPERATION=RELEASE for the first subcommand. Then when you begin the second subcommand, the DFSMSrmm API module reuses resources, such as work areas, that it obtained for the first subcommand. Reusing resources can reduce processing overhead associated with releasing and obtaining resources.

If you do not use OPERATION=RELEASE, when the second subcommand request starts, all data and pointers for the first subcommand are overwritten.

Note: You do not specify SUBCMDADDR for OPERATION=CONTINUE, and you do not specify SUBCMDADDR or OUTBUFADDR for OPERATION=RELEASE.

Single Parameter List, Multiple Token Areas

When you use a single parameter list and multiple token areas:

- More than one subcommand request can be active at a time.
- Only one subcommand request can be processed at any given time.

This variation allows you to continue a previous subcommand after you have started another. You might need to use multiple token areas when your application program is designed to support a sequence of subcommand requests like the one that follows:

1. Use a SEARCHVOLUME subcommand to request volume information. For example:

   ```
   SEARCHVOLUME OWNER(userid) LIMIT(*)
   ```

2. Use a SEARCHDATASET subcommand to obtain data set information. For example:

   ```
   SEARCHDATASET VOLUME(volser) LIMIT(*)
   ```

3. Repeat subcommands until all information for all data sets is obtained and passed back to your user.

Figure 10 shows the use of a single parameter list and multiple tokens to identify work areas. The multiple token areas allow the flexibility of continuing a previous subcommand after starting another subcommand. Use the token you obtained from the previous subcommand when you want to continue that subcommand.

```
****************************************
** Start the first subcommand
****************************************
  XC    TOKEN1,TOKEN1            No resources/token yet
  LA    R4,SUBCMD1               Point to 1st subcommand
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=BEGIN,                X
        TOKEN=TOKEN1,                                   X
        SUBCMDADDR=(R4),OUTBUFADDR=(R3)
  ...
****************************************
** Start the second subcommand
****************************************
  XC    TOKEN2,TOKEN2            No resources/token yet
  LA    R4,SUBCMD2               Point to 2nd subcommand
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=BEGIN,                X
        TOKEN=TOKEN2,                                   X
        SUBCMDADDR=(R4),OUTBUFADDR=(R3)
  ...
```

Figure 10. (Part 1 of 2) Single Parameter List, Multiple Token Areas

```
****************************************
** Continue the second subcommand
****************************************
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                   X
        APIADDR=APIMOD@,OPERATION=CONTINUE,            X
        TOKEN=TOKEN2,                                  X
        OUTBUFADDR=(R3)
  ...
****************************************
** Continue the first subcommand
****************************************
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                   X
        APIADDR=APIMOD@,OPERATION=CONTINUE,            X
        TOKEN=TOKEN1,                                  X
        OUTBUFADDR=(R3)
  ...
****************************************
** Release resources for the first subcommand
****************************************
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                   X
        APIADDR=APIMOD@,OPERATION=RELEASE,             X
        TOKEN=TOKEN1
  ...
****************************************
** Release resources for the second subcommand
****************************************
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                   X
        APIADDR=APIMOD@,OPERATION=RELEASE,             X
        TOKEN=TOKEN2
```

Figure 11. (Part 2 of 2) Single Parameter List, Multiple Token Areas

Figure 10 on page 19 shows how you can reuse resources. When your application program is finished with the first subcommand request, it can reuse the first token to begin a third request. Note, however, that once a token is reused to begin a new subcommand request, you cannot continue the previous request associated with that token.

In Figure 10 on page 19, the same output buffers are used for all subcommand requests. As a result, all of the output data in the output buffer must be processed before another request can be started or continued. To avoid this situation, you might write your application program to use multiple output buffers instead of a single output buffer.

Figure 10 on page 19 shows multiple releases using the OPERATION=RELEASE parameter. Instead of using multiple releases, you can specify the OPERATION=ENDALL once to free all resources associated with all tokens. See Figure 12 on page 21 for an example of this method.

Note: You do not specify the TOKEN parameter when you use OPERATION=ENDALL. Your application program, however, is responsible for setting all tokens to zeros to prevent them from being reused.

```
****************************************
** Release all resources
****************************************
  EDGXCI  MF=(E,PLIST),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=ENDALL
```

Figure 12. Releasing All Resources

Your application program might encounter a resource constraint condition like short-on-storage before it issues the OPERATION=ENDALL.

Multiple Parameter List, Single Token Area

When you use multiple parameter lists with a single token:

- Only one subcommand can be active at a time.
- Different parameter lists can be used to:
  - Begin subcommand requests.
  - Continue subcommand requests.
  - Release resources.
- Starting a new subcommand request ends any previous subcommand request.

Figure 13 on page 22 shows the use of multiple parameter lists and a single token area. With a single token area, you cannot continue the first subcommand request, even though there are multiple parameter lists. The variation in Figure 13 on page 22 prevents you from continuing the first subcommand after you begin the second subcommand.

```
*****************************************
** Start the first subcommand
*****************************************
  XC    TOKENA,TOKENA            No resources/token yet
  LA    R4,SUBCMD1               Point to 1st subcommand
  EDGXCI  MF=(E,BEGINPL),PLISTVER=MAX,                  X
        APIADDR=APIMOD@,OPERATION=BEGIN,                X
        TOKEN=TOKENA,                                   X
        SUBCMDADDR=(R4),OUTBUFADDR=(R3)
  ...
*****************************************
** Continue the subcommand
*****************************************
  EDGXCI  MF=(E,CONTPL),PLISTVER=MAX,                   X
        APIADDR=APIMOD@,OPERATION=CONTINUE,             X
        TOKEN=TOKENA,                                   X
        OUTBUFADDR=(R3)
  ...
*****************************************
** Done with the subcommand, release
*****************************************
  EDGXCI  MF=(E,RELPL),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=RELEASE,              X
        TOKEN=TOKENA
  ...
*****************************************
** Start the second subcommand
*****************************************
  LA    R4,SUBCMD2               Point to 2nd subcommand
  EDGXCI  MF=(E,BEGINPL),PLISTVER=MAX,                  X
        APIADDR=APIMOD@,OPERATION=BEGIN,                X
        TOKEN=TOKENA,                                   X
        SUBCMDADDR=(R4),OUTBUFADDR=(R3)
*****************************************
** Continue the subcommand
*****************************************
  EDGXCI  MF=(E,CONTPL),PLISTVER=MAX,                   X
        APIADDR=APIMOD@,OPERATION=CONTINUE,             X
        TOKEN=TOKENA,                                   X
        OUTBUFADDR=(R3)
  ...
*****************************************
** Done with the subcommand, release
*****************************************
  EDGXCI  MF=(E,RELPL),PLISTVER=MAX,                    X
        APIADDR=APIMOD@,OPERATION=RELEASE,              X
        TOKEN=TOKENA
```

Figure 13. Multiple Parameter Lists, Single Token Area

Multiple Parameter List, Multiple Token Areas

When you use multiple parameter lists and multiple token areas:

- More than one subcommand request can be active at a time.
- More than one active subcommand request can be processed at a time.
- Different parameter lists can be used to:
  - Begin subcommand requests.
  - Continue subcommand requests.
  - Release resources.

This variation lends itself to processing in re-entrant code where subroutines can be created for commonly used code. Figure 14 on page 23 shows how the same

subroutines can be used to issue and process multiple subcommand requests with each having its own token and output buffer area.

```
****************************************
** Start the first subcommand
****************************************
  XC    TOKENA,TOKENA            No resources/token yet
  LA    R2,TOKENA                Point to 1st token
  LA    R3,OUTBUF1               Point to 1st buffer
  LA    R4,SUBCMD1               Point to 1st subcommand
  BAS   R9,BEGRTN                Issue command
  ...
****************************************
** Start the second subcommand
****************************************
  LA    R2,TOKENB                Point to 2nd token
  LA    R3,OUTBUF2               Point to 2nd buffer
  LA    R4,SUBCMD2               Point to 2nd subcommand
  BAS   R9,BEGRTN                Issue command
  ...
```

Figure 14. (Part 1 of 2) Multiple Parameter Lists, Multiple Token Area

```
      ****************************************
      ** Continue the 2nd subcommand
      ****************************************
        LA    R2,TOKENB               Point to 2nd token
        BAS   R9,CONRTN               Continue 2nd cmd
        ...
      ****************************************
      ** Continue the 1st subcommand
      ****************************************
        LA    R2,TOKENA               Point to 1st token
        BAS   R9,CONRTN               Continue 1st cmd
        ...
      ****************************************
      ** Done with the subcommands, release
      ****************************************
        LA    R2,TOKENA               Point to 1st token
        BAS   R9,RELTRN               Release 1st token
        ...
        LA    R2,TOKENB               Point to 2nd token
        BAS   R9,RELTRN               Release 2nd token
        ...
BEGRTN  EQU   *
        EDGXCI  MF=(E,BEGINPL),PLISTVER=MAX,                  X
              APIADDR=APIMOD@,OPERATION=BEGIN,                X
              TOKEN=(R2),                                     X
              SUBCMDADDR=(R4),OUTBUFADDR=(R3)
        BR    R9
        ...
CONRTN  EQU   *
      ****************************************
      ** Continue the subcommand
      ****************************************
        EDGXCI  MF=(E,CONTPL),PLISTVER=MAX,                   X
              APIADDR=APIMOD@,OPERATION=CONTINUE,             X
              TOKEN=(R2),                                     X
              OUTBUFADDR=(R3)
        BR    R9
        ...
RELRTN  EQU   *
      ****************************************
      ** Done with the subcommand, release
      ****************************************
        EDGXCI  MF=(E,RELPL),PLISTVER=MAX,                    X
              APIADDR=APIMOD@,OPERATION=RELEASE,              X
              TOKEN=(R2)
        BR    R9
```

Figure 15. (Part 2 of 2) Multiple Parameter Lists, Multiple Token Area

Freeing Resources

When you begin a new subcommand request, the DFSMSrmm API either obtains a new set of resources when you provide a token set to all zeros, or reuses resources associated with a valid, nonzero token that you provide.

The DFSMSrmm API does not free any resources or clear any tokens when it completes processing a subcommand request. Resources are freed under these conditions.

- You specify the OPERATION=RELEASE and TOKEN=token parameters to free all resources associated with the specified token as shown in Figure 16 on page 25.

```
************************************************
** Done with the subcommand, setup release parmlist
************************************************
  EDGXCI  MF=(M,RELPL,NOCHECK),PLISTVER=MAX,            X
        APIADDR=APIMOD@,OPERATION=RELEASE

************************************************
** Call the DFSMSrmm API
************************************************
  EDGXCI  MF=(E,RELPL,NOCHECK),TOKEN=TOKENA
```

Figure 16. TOKEN= Specified on EDGXCI

Specifying TOKEN=TOKENA on the execute form of EDGXCI causes the 4-byte TOKENA area to be set to all zeros upon return from freeing the token.

TOKEN=token is required even when you specify MF=(E,label,NOCHECK), unless you also specify OPERATION=ENDALL. Specifying TOKEN=token causes the 4-byte token area to be updated upon return from the DFSMSrmm API. The token is set to all zeros by the EDGXCI macro expansion.

- You specify the OPERATION=ENDALL parameter to free all resources associated with all tokens, as shown in Figure 17.

  Note: You are responsible for setting applicable tokens to all zeros when you specify OPERATION=ENDALL.
- Your application program ends (end-of-task occurs).

To release a resource, you need to have access to the tokens associated with the resources you want to release. If you no longer have access to the tokens or you have set the tokens to all zeros before you use OPERATION=RELEASE, there are only two ways you can free the resources.

- Your application program specifies OPERATION=ENDALL to free all resources associated with all tokens.
- Your application program ends (end-of-task occurs).

In Figure 17, the OPERATION=ENDALL parameter is specified and TOKEN is not required.

```
************************************************
** Done with the subcommand, setup endall parmlist
************************************************
  EDGXCI  MF=(M,RELPL,NOCHECK),PLISTVER=MAX,            X
        APIADDR=APIMOD@

************************************************
** Call the DFSMSrmm API
************************************************
  EDGXCI  MF=(E,RELPL,NOCHECK),OPERATION=ENDALL
```

Figure 17. TOKEN= Not Specified on EDGXCI

Chapter 3. Receiving Output Data in the Output Buffer

The DFSMSrmm application programming interface returns data in the output buffer you define. The data is in the following format:

- A four-byte length field into which your application program sets the total size of the output buffer.
- A four-byte length field that is used by DFSMSrmm when your output buffer is too small.
- A four-byte length field containing the total size of all the output including the bytes of the length field.
- Structured fields which consist of structured field introducers (SFI) and data.
  - An SFI is a structure that separates one line or field of output data from another. SFIs are described in “Description of Structured Fields”.
  - Data in line format or field format.

Use the EDGXSF macro described in “EDGXSF: Structured Field Definitions” on page 76 to map the output buffer header and the structured field introducers. EDGXSF also defines values used in the output fields. Do not hardcode the offsets because they might change in the future.

Note: The DFSMSrmm API does not return data for all the subcommands you specify in your request. See “SFIs for Output Data for Subcommands” on page 37 for information on subcommands which result in structured field output in your buffer.

Description of Structured Fields

A structured field consists of:

- A Structured Field Introducer (SFI)
- Data that follows the SFI as described below:

Part **Description**

SFI Structured Field Introducer. A structure with a minimum size of 8 bytes in the following format:

Byte count	Description
2	Two-byte length. The length includes the length of the SFI (8 bytes) and the length of the data following the SFI.
3	Three-byte SFI identifier (ID)
1	One-byte SFI type modifier
1	One-byte (reserved)
1	One-byte data-type identifier

Data Data following the SFI which can contain actual data, no data, binary zeros, or blank data.

See “Appendix A. Structured Field Introducers” on page 55 for descriptions of the SFIs that the DFSMSrmm API returns.

Structured fields can appear in any order. Write your application so it skips over any structured field it is not prepared to handle. This makes your application program less sensitive to changes like enhancements to DFSMSrmm that introduce new or different structured fields and sequences. You can update your application program when it is convenient to do so rather than being forced to do so because your application program no longer works.

We use the convention <SFI>data to describe the SFIs in the examples that follow. <SFI>data denotes a Structured Field Introducer (SFI) that is followed by data. In the examples, we replace the term “SFI” with its descriptive name, for example: <data-set-name>. There is no association between the length of a particular SFI and its descriptive name.

SFI Data Format

You determine if the DFSMSrmm API returns line format or field format data to your application program. Line format is where fixed text and variable data are formatted into lines suitable for displaying at a terminal or for printing, while field format is where the output consists only of SFIs and variable data.

You can request that the data be returned in line format when you specify the EDGXCI macro OUTPUT=LINES parameter. You can request that the data be returned in field format by specifying the OUTPUT=FIELDS parameter.

When you specify the EDGXCI macro OUTPUT=LINES parameter, the DFSMSrmm API returns the output lines in the same format as information returned by the DFSMSrmm RMM TSO subcommand.

In the examples that follow, assume that there is only one data set on volume VOL001: OWNERONE.FIELD.TEST.

Requesting Line Format

Figure 18 on page 29 is an example of the line format data that the DFSMSrmm API returns when you specify the OUTPUT=LINES parameter. Your request specifying LISTDATASET FIELD.TEST VOLUME(VOL001) subcommand might produce the output that is shown in Figure 18 on page 29.

```
<Begin DATASET group>
 <line>Data set name = OWNERONE.FIELD.TEST
 <line>Job name      = TESTAPI
 <line>Volume        = VOL001               Physical file sequence number = 1
 <line>Device number = 0BE0       Owner       = OWNERONE Data set sequence = 1
 <line>Create date   = 1997/04/27 Create time = 07:41:29 System ID         = 9021
 <line>Block size    = 80         Block count = 5   Total block count      = 900
 <line>Percent of volume  =    23
 <line>Logical Record Length = 80         Record Format  = FB
 <line>Date last written     =            Date last read = 1997/04/30
 <line>Step name             = STEP01     DD name        = OUTPUT
 <line>Management class      =            VRS management value  =
 <line>Storage group         = STG0S1     VRS retention date    =
 <line>Storage class         = SCFAST     VRS retained          = NO
 <line>Data class            = DCCLS1     ABEND while open      = NO
 <line>                                   Catalog status        = UNKNOWN
 <line>Primary VRS details:
 <line>       Name           = TESTA.**
 <line>       Job name       =            Type                  = DATASET
 <line>       Subchain NAME  = VRS1       Subchain start date   = 1997/23
 <line>Secondary VRS details:
 <line>       Value or class = MV*
 <line>       Job name       =
 <line>       Subchain NAME  = M2         Subchain start date   = 1997/24
 <line>Security class        =       Description   =
<End DATASET group>
```

Figure 18. Example of List Type of Output Using OUTPUT=LINES

Note: <line> is the Structured Field Introducer for each line and is followed by the data returned from specifying the RMM LISTDATASET subcommand.

Requesting Field Format

Figure 19 is an example of the field format data that the DFSMSrmm API returns when you specify the OUTPUT=FIELDS parameter. Your request specifying LISTDATASET FIELD.TEST VOLUME(VOL001) subcommand might also produce the output shown in Figure 19.

```
<Begin DATASET group>
  <DSN  - data set name            : 44 , character:         >
  <CJBN - job name                 : 8  , character:         >
  <VOL  - volume serial            : 6  , character:         >
  <OWN  - owner                    : 8  , character:         >
  <DSEQ - data set sequence        : 4  , bin(31):           >
  <DEV  - device number (address)  : 4  , hexadecimal:       >
  <FILE - physical file sequence   : 4  , bin(31):           >
  <CDTJ - create date              : 4  , packed decimal:    >
  <CTM  - create time              : 4  , packed decimal:    >
  <SYS  - SMF system id            : 8  , character:         >
  <BLKS - block size               : 4  , bin(31):           >
  <BLKC - block count              : 4  , bin(31):           >
  <LRCL - logical record length    : 4  , bin(31):           >
  <RCFM - record format            : 4  , character:         >
  <DC   - data class               : 8  , character:         >
  <DLWJ - date last written        : 4  , packed decimal:    >
  <DLRJ - date last read           : 4  , packed decimal:    >
```

Figure 19. (Part 1 of 2) Example of Output Using OUTPUT=FIELDS

```
  <STEP - step name                : 8  , character:         >
  <DD   - dd name                  : 8  , character:         >
  <MC   - management class         : 8  , character:         >
  <SG   - storage group name       : 8  , character:         >
  <SC   - storage class            : 8  , character:         >
  <VMV  - VRS management value     : 8  , character:         >
  <RTDJ - retention date           : 4  , packed decimal:    >
  <VTYP - Primary VRS type         : 1  , bin(8):            >
  <VJBN - Primary VRS jobn         : 8  , character:         >
  <VNME - Primary VRS name         : 44 , character:         >
  <VSCN - Primary VRS subchain name: 8  , character:         >
  <VSCD - Primary VRS subchain date: 4  , packed decimal:    >
  <VRSR - VRS retained             : 1  , bin(8):            >
  <NME  - security class name      : 8  , character:         >
  <CLS  - sec class description    : 32 , character:         >
  <ABND - Abend while open         : 1  , bin(8):            >
  <CTLG - Catalog status           : 1  , bin(8):            >
  <2JBN - Secondary VRS jobnme mask: 8  , character:         >
  <2NME - Secondary VRS mask       : 8  , character:         >
  <2SCN - Second. VRS subchain name: 8  , character:         >
  <2SCD - Second. VRS subchain date: 4  , packed decimal:    >
  <BLKT - Total block count        : 4  , bin(31):           >
  <CPGM - Creating program name    : 8  , character:         >
  <LPGM - Last used program name   : 8  , character:         >
  <LJOB - Last used job            : 8  , character:         >
  <LSTP - Last used step name      : 8  , character:         >
  <LDD  - Last used DD name        : 8  , character:         >
  <LDEV - Last drive               : 4  , character:         >
  <DPCT - Percent of volume        : 1  , bin(8):            >
<End DATASET group>
```

Figure 20. (Part 2 of 2) Example of Output Using OUTPUT=FIELDS

Figure 19 on page 29:

- Shows begin and end group SFIs. In this example, <Begin DATASET Group> and <End DATASET Group>.
- Includes descriptive names used to identify Structured Field Introducers. The SFI identifies the data type; and the long character <...> strings do not represent the actual size of the SFIs, which are only 8 bytes in length.
- Can appear to have no data. This is because structured fields can
  - Have no data (SFI only, as in this example), binary zeros, or blank characters.
  - Be omitted if they have no data.
- Shows that structured fields can be order independent. For example, VOL in Figure 38 on page 43 occurs before OWN for LISTDATASET while OWN occurs before VOL for LISTPRODUCT in Figure 41 on page 44.
- Shows that structured fields might not be in the same order as their corresponding positions in any line-format output.
- Shows variable-length fields.

Refer to “Appendix D. Hexadecimal Example of an Output Buffer” on page 81 for an example of an output buffer in hexadecimal representation.

Types of Output

The DFSMSrmm API can produce standard output and expanded output depending on the values you specify for the OUTPUT and EXPAND parameters as described in “EDGXCI Parameters” on page 5.

The examples shown in “Requesting Standard Output” and “Requesting Expanded Output”:

- Assume that there is only one data set on volume VOL001: OWNERONE.FIELD.TEST.
- Use SFI data type descriptions, such as DSN for data set name.
- Show maximum length values, without the term “bytes”.
- Show the data type, such as character.

Requesting Standard Output

When you specify EXPAND=NO, your request specifying the SEARCHDATASET VOLUME(VOL001) subcommand might produce the output that is shown in Figure 21.

```
<Begin DATASET group>
  <DSN  - data set name  : 44 , character:      >OWNERONE.FIELD.TEST
  <VOL  - volume serial  : 6  , character:      >VOL001
  <OWN  - owner          : 8  , character:      >OWNERONE
  <CDTJ - create date    : 4  , packed decimal: >x'1997117C'
  <CTM  - create time    : 4  , packed decimal: >x'0815270C'
  <FILE - phys file seq  : 4  , bin(31):        >x'00000001'
  <End DATASET group>
```

Figure 21. Example of Search Type of Output Using EXPAND=NO

Refer to “Appendix D. Hexadecimal Example of an Output Buffer” on page 81 for a hexadecimal representation and discussion of the contents of the output buffer shown in Figure 21.

Requesting Expanded Output

The DFSMSrmm API can provide expanded output for the DFSMSrmm TSO RMM SEARCHDATASET, SEARCHPRODUCT, SEARCHVOLUME, and SEARCHVRS subcommands when you specify OUTPUT=FIELDS and EXPAND=YES or use the default EXPAND=YES in your application program.

The DFSMSrmm API does not provide expanded data for the DFSMSrmm TSO RMM SEARCHBIN or SEARCHRACK subcommands.

When you specify OUTPUT=FIELDS and EXPAND=YES, your SEARCHDATASET VOLUME(VOL001) subcommand might produce the output that is shown in Figure 22 on page 32.

```
<Begin DATASET Group>
  <DSN  - data set name         : 44 , character:      >OWNERONE.FIELD.TEST
  <CJBN - job name              : 8  , character:      >TESTAPI
  <VOL  - volume serial         : 6  , character:      >VOL001
  <OWN  - owner                 : 8  , character:      >OWNERONE
  <DSEQ - data set sequence     : 4  , bin(31):        >x'00000001'
  <DEV  - device number         : 4  , bin(31):        >0BE0
  <FILE - physical file seq     : 4  , bin(31):        >x'00000001'
  <CDTJ - create date           : 4  , packed decimal: >x'1997117C'
  <CTM  - create time           : 4  , packed decimal: >x'0741290C'
```

Figure 22. (Part 1 of 2) Example of Search Type of Output Using OUTPUT=FIELDS, EXPAND=YES

```
  <SYS  - SMF system id         : 8  , character:      >9021
  <BLKS - block size            : 4  , bin(31):        >x'00000050'
  <BLKC - block count           : 4  , bin(31):        >x'00000005'
  <LRCL - logical rcd length    : 4  , bin(31):        >x'00000050'
  <RCFM - record format         : 4  , character:      >FB
  <DC   - data class            : 8  , character:      >DCCLS1
  <DLWJ - date last written     : 4  , packed decimal: >
  <DLRJ - date last read        : 4  , packed decimal: >1997117F
  <STEP - step name             : 8  , character:      >STEP01
  <DD   - dd name               : 8  , character:      >OUTPUT
  <MC   - management class      : 8  , character:      >
  <SG   - storage group         : 8  , character:      >STG0S
  <SC   - storage class         : 8  , character:      >SCFAST
  <VMV  - VRS value             : 8  , character:      >
  <RTDJ - VRS retention date    : 4  , packed decimal: >
  <VTYP - Primary VRS type      : 1  , bin(8):         >1
  <VJBN - Primary VRS jobn      : 8  , character:      >TEST*
  <VNME - Primary VRS name      : 44 , character:      >TESTA.**
  <VSCN - Primary VRS s-chain n.: 8  , character:      >VRS1
  <VSCD - Primary VRS s-chain d.: 4  , packed decimal: >1997/230
  <VRSR - VRS retained          : 1  , bin(8):         >1
  <NME  - security class        : 8  , character:      >
  <CLS  - secl description      : 32 , character:      >
  <ABND - Abend while open      : 1  , bin(8):         >0
  <CTLG - Catalog status        : 1  , bin(8):         >0
  <2JBN - Seco. VRS jobname mask: 8  , character:      >
  <2NME - Secondary VRS mask    : 8  , character:      >MV*
  <2SCN - Second. VRS s-chain n.: 8  , character:      >M2
  <2SCD - Second. VRS s-chain d.: 4  , packed decimal: >1997/241
  <BLKT - Total block count     : 4  , bin(31):         >
<End DATASET Group  >
```

Figure 23. (Part 2 of 2) Example of Search Type of Output Using OUTPUT=FIELDS, EXPAND=YES

Return and Reason Codes

DFSMSrmm returns return codes and reason codes to your application program in the general purpose registers and also as data in your output buffer as follows:

- Return codes and reason codes issued as a result of processing of your subcommand request. Refer to *OS/390 DFSMSrmm Guide and Reference* for information about these codes.
- Return codes and reason codes associated with the API itself. These are the return codes and reason codes listed in “EDGXCI Return and Reason Codes” on page 9 for macro EDGXCI.

- Return and reason codes from system services. DFSMSrmm uses various system services, such as catalog services, to process the subcommands from your application program. When DFSMSrmm receives a non-zero return code from a system service, the DFSMSrmm API places the return code and associated reason code in your output buffer as structured fields, along with a name to identify the service. See “System Return and Reason Code SFIs” on page 35 for more information.

Messages and Message Variables

The DFSMSrmm API can return messages and message variables in your output buffer. Figure 24 show how messages are returned in line format when you specify the OUTPUT=LINES parameter and field format when you specify the OUTPUT=FIELDS parameter.

```
  <message line>message text
  <message line>message text

or

  <Begin MESSAGE group>
    <message number  >number
    <message variable>variable
  <End MESSAGE group>
  <Begin MESSAGE group>
    <message number  >number
    <message variable>variable
  <End MESSAGE group>
```

Figure 24. Message and Message Variable Structured Fields. Message and Message Variable Structured Fields

Refer to “Messages and Message Variables SFIs” on page 36 for information about which messages can be placed in your output buffer.

Date Format and Time Format

DFSMSrmm dates are in packed decimal format: yyyydddC, where yyyyddd is a Julian date and C is a standard packed-decimal sign character. The date formats used are returned in internal format and can be interpreted as follows:

- Interpret 9999366 as PERMANENT retention date format.
- Interpret 9999365 as PERMANENT retention date format.
- Interpret 9800000 as WHILECATLG retention date format.
- Interpret 98ccccc as CYCL/ccccc retention date format.
- Interpret 0000098 as CATRETPD retention date format.
- Interpret yyyyddd as yyyy/mm/dd, yyyy/dd/mm, mm/dd/yyyy, dd/mm/yyyy, dd/yyyy/mm, mm/yyyy/dd.

DFSMSrmm also returns time in packed decimal format: hhmmsstC, where hhmmsst is the time in hours, minutes, seconds, and tenths of seconds and C is a standard packed-decimal sign character.

Identifying Structured Field Introducers

A structured field introducer (SFI) is a structure that identifies one line or field of output data from another. The DFSMSrmm API returns these types of SFIs in your output buffer:

- SFIs that begin and end a resource group as described in "Begin and End Resource Groups".
- SFIs that introduce a single line of output data as described in:
  - "System Return and Reason Code SFIs" on page 35
  - "Messages and Message Variables SFIs" on page 36
  - "Add Type of Subcommands" on page 38
  - "Change Type of Subcommands" on page 38
  - "Delete Type of Subcommands" on page 38
  - "GETVOLUME Subcommand" on page 39
  - "List Type of Subcommands" on page 39
  - "Search Type of Subcommands" on page 47

The following notation indicates an SFI:

```
<xxxx - descriptive name       : data length, data type :  >
```

where "xxxx" is a character type of mnemonic. In your application program, you need to use the 3-byte hexadecimal identifiers for Structured Field Introducers.

"Appendix A. Structured Field Introducers" on page 55 describes all the structured fields that the DFSMSrmm API can return to your application program.

Note: The DFSMSrmm API does not return information for all subcommands. For example, the DFSMSrmm API does not produce structured fields for a successful ADDBIN subcommand request.

Begin and End Resource Groups

In the previous examples, you saw that output structured fields were grouped by a pair of unique Structured Field Introducers as shown in Figure 25.

```
<Begin DATASET group>
  <..         >data set name
  <..         >volume id
<End DATASET group>
```

Figure 25. Begin and End Resource Group SFI Sequence. This shows the Begin and End Resource Groups that group output data structured fields.

The begin and end resource group SFIs identify when output for a particular resource, such as a data set, begins and ends. The pairs of Begin and End Resource Group SFIs are shown in Figure 26 on page 35.

```
<Begin BIN group>       <End BIN group>
<Begin CONTROL group>   <End CONTROL group>
<Begin DATASET group>   <End DATASET group>
<Begin MESSAGE group>   <End MESSAGE group>
<Begin OWNER group>     <End OWNER group>
<Begin PRODUCT group>   <End PRODUCT group>
<Begin RACK group>      <End RACK group>
<Begin VOLUME group>    <End VOLUME group>
<Begin VRS group>       <End VRS group>
```

Figure 26. Begin and End Resource Group SFI Pairs. This shows the pairs of Begin and End Resource Group SFIs used to enclose output data structured fields.

In addition to identifying the beginning and ending of output for a particular resource, the Begin and End Resource Group SFIs shown in Figure 27 are used to differentiate one subgroup of data from another in the output the DFSMSrmm API returns for the LISTCONTROL, LISTVOLUME, and SEARCHVOLUME subcommands.

```
<Begin ACCESS group>   <End ACCESS group>
<Begin ACTIONS group>  <End ACTIONS group>
<Begin CNTL group>     <End CNTL group>
<Begin LOCDEF group>   <End LOCDEF group>
<Begin MNTMSG group>   <End MNTMSG group>
<Begin MOVES group>    <End MOVES group>
<Begin OPTION group>   <End OPTION group>
<Begin REJECT group>   <End REJECT group>
<Begin SECCLS group>   <End SECCLS group>
<Begin STATS group>    <End STATS group>
<Begin STORE group>    <End STORE group>
<Begin VLPOOL group>   <End VLPOOL group>
<Begin VOL group>      <End VOL group>
```

Figure 27. Begin and End Resource Group SFI Pairs for Subgroups. This shows the pairs of Begin and End Resource Group SFIs used to differentiate subgroups of data.

Groups and subgroups, such as MESSAGE and SECCLS, are repeated as often as necessary to differentiate resources.

System Return and Reason Code SFIs

When DFSMSrmm receives a non-zero return code from a system service, the system return code and associated reason code are put into your output buffer as shown in Figure 28. DFSMSrmm issues return code 116 and reason code 06 when an error like this occurs.

```
<Begin SYSRETC group>
  <SVCN - service name             : 16 , character:         >
  <RTNC - return code              : 4  , bin(31):           >
  <RSNC - reason code              : 4  , bin(31):           >
<End SYSRETC group>
```

Figure 28. System Return and Reason Codes. This shows the sequence of SFIs used when a non-zero return code is received by DFSMSrmm from a call to a system service.

The DFSMSrmm API returns the same SFIs for both line format and field format.

Messages and Message Variables SFIs

When messages or message variables are returned to you as output data, they are put into your output buffer as structured fields as shown in Figure 29.

```
    <MSGL - message line             : nn , character:         >
    <MSGL - message line             : nn , character:         >

or

  <Begin MESSAGE group>
    <MSGN - message number           : 8  , character:         >
    <xxx  - variable>
  <End MESSAGE group>
  <Begin MESSAGE group>
    <MSGN - message number           : 8  , character:         >
    <xxx  - variable>
  <End MESSAGE group>
```

Figure 29. SFIs for Messages and Message Variables

When you specify OUTPUT=LINES, messages issued by DFSMSrmm are placed in your output buffer using the LINE SFI.

When you specify OUTPUT=FIELDS, only the messages listed in Table 5 are placed in your output buffer. These messages, some of which are issued only in conjunction with a subcommand parameter such as POOL or COUNT, are included in the output because they contain data and codes that can be especially useful to your application. Your application program should use the return and reason codes that it receives rather than messages to determine whether or not the subcommand request successful.

Table 5 lists:

- The Structured Field Introducers that follow the <MSGN> SFI
- The applicable subcommands
- A non-inclusive list of the return codes (RC) and reason codes (RSN).

Table 5. Message Related SFIs

Message	SFI ID(s)	Subcommand(s)	RC	RSN(s)
EDG3010	ENTN	All SEARCH subcommands when no (0) entry is returned	4	8
EDG3011	ENTN	All SEARCH subcommands when 1 entry returned	0 4	0 2 and 4
EDG3012	ENTN	All SEARCH subcommands when > 1 entry returned	0 4	0 2 and 4
EDG3013	VOL	AV	12	many
EDG3014	CNT	AV	12	many
EDG3015	OWN VOL	GV	0	0
EDG3016	RCK	AV CV	0	0
EDG3017	RCK	AB AR	12	18 68 70
EDG3018	CNT	AB AR	12	18 68 70

Table 5. Message Related SFIs (continued)

Message	SFI ID(s)	Subcommand(s)	RC	RSN(s)
EDG3019	RCK	DB DR	12	many
EDG3020	CNT	DB DR	12	many
EDG3277	FRC FRS	AV CV	12	122
EDG3278	CSG	AV CV	12	124
EDG3288	FRC FRS VOL	CV DV	12	132
EDG3289	FRC FRS	CV	12	134
EDG3292	CLIB	AV CV	12	140
EDG3301	FRC FRS	AV CV GV	12	152
EDG3310	CLIB	CV DV	12	170
EDG3311	FRC FRS	AV CV DV	12	172
EDG3314	MEDN	CV	12	176
EDG3328	KEYF KEYT TYPF TYPT	SD SV	4	12

See *OS/390 MVS System Messages, Vol 2 (ASB-EZM)* for the DFSMSrmm messages and *OS/390 DFSMSrmm Guide and Reference* for DFSMSrmm return and reason codes.

SFIs for Output Data for Subcommands

The DFSMSrmm API returns various types of output to your application program:

- Return and reason codes in registers from DFSMSrmm and the DFSMSrmm API.
- Return and reason codes from system services in structured fields.
- List header lines as formatted lines in structured fields.
- Messages as formatted lines or as message variables in structured fields.
- Report output data as formatted lines or as unformatted fields in structured fields.

When you specify OUTPUT=LINES, the DFSMSrmm API returns output data, except for system return and reason codes, as formatted lines in structured fields. The structured fields are introduced by the <LINE> and <MSGL> Structured Field Introducers as shown in Figure 30. DFSMSrmm places system return codes and reason codes in your output buffer as described in “System Return and Reason Code SFIs” on page 35.

```
<Begin resource group>
  <LINE - Formatted output line   : nn , character:      >
  <LINE - Formatted output line   : nn , character:      >
  <MSGL - Formatted output message: nn , character:      >
  <MSGL - Formatted output message: nn , character:      >
<End resource group>
```

Figure 30. Formatted Lines

When you specify OUTPUT=FIELDS, the DFSMSrmm API returns output data as unformatted fields in structured fields with or without data.

Add Type of Subcommands

The DFSMSrmm Add type of subcommands are: ADDBIN, ADDDATASET, ADDOWNER, ADDPRODUCT, ADDRACK, ADDVOLUME, and ADDVRS. You use these subcommands to add information to the DFSMSrmm control data set.

The DFSMSrmm API returns information when:

- You specify the ADDVOLUME subcommand with the POOL operand. The DFSMSrmm API returns the rack number that is assigned to the volume as output data as shown in Figure 31.
- When an error occurs, and then only for specific return and reason code combinations described “Messages and Message Variables SFIs” on page 36 and “SFIs for Return and Reason Codes” on page 56.

```
<Begin VOLUME group>
  <Begin MESSAGE group>
    <MSGN - message number           : 8  , character:         >
    <RCK  - rack or bin number       : 6  , character:         >
  <End MESSAGE group>
<End VOLUME group>
```

Figure 31. SFIs for ADDVOLUME with OUTPUT=FIELDS

Change Type of Subcommands

The DFSMSrmm Change type of subcommands are: CHANGEDATASET, CHANGEOWNER, CHANGEPRODUCT,and CHANGEVOLUME. You use these subcommands to change information in the DFSMSrmm control data set.

The DFSMSrmm API returns information when:

- You specify the CHANGEVOLUME subcommand with the POOL operand. The DFSMSrmm API returns the rack number that is assigned to the volume as output data as shown in Figure 32.
- When an error occurs, and then only for specific return and reason code combinations described “Messages and Message Variables SFIs” on page 36 and “SFIs for Return and Reason Codes” on page 56.

```
<Begin VOLUME group>
  <Begin MESSAGE group>
    <MSGN - message number           : 8  , character:         >
    <RCK  - rack or bin number       : 6  , character:         >
  <End MESSAGE group>
<End VOLUME group>
```

Figure 32. SFIs for CHANGEVOLUME with OUTPUT=FIELDS

Delete Type of Subcommands

The DFSMSrmm Delete type of subcommands are: DELETEBIN, DELETEDATASET, DELETEOWNER, DELETEPRODUCT, DELETERACK, DELETEVOLUME, and DELETEVRS. You use these subcommands to delete information from the DFSMSrmm control data set.

The DFSMSrmm API returns information when an error occurs, and then only for specific return and reason code combinations described “Messages and Message Variables SFIs” on page 36 and “SFIs for Return and Reason Codes” on page 56.

GETVOLUME Subcommand

You use the RMM GETVOLUME subcommand to obtain a volume from DFSMSrmm.

The DFSMSrmm API returns information when:

- The GETVOLUME request was successful. The DFSMSrmm API returns volume information and owner information as shown in Figure 33.
- When an error occurs, and then only for specific return and reason code combinations described “Messages and Message Variables SFIs” on page 36 and “SFIs for Return and Reason Codes” on page 56.

```
<Begin VOLUME group>
  <Begin MESSAGE group>
    <MSGN - message number           : 8  , character:         >
    <VOL  - volume serial            : 6  , character:         >
    <OWN  - owner                    : 8  , character:         >
  <End MESSAGE group>
<End VOLUME group>
```

Figure 33. SFIs for GETVOLUME with OUTPUT=FIELDS

List Type of Subcommands

The DFSMSrmm List type of subcommands are: LISTBIN, LISTCONTROL, LISTDATASET, LISTOWNER, LISTPRODUCT, LISTRACK, LISTVOLUME, and LISTVRS. You use these subcommands to obtain information from the DFSMSrmm control data set about a single resource.

The DFSMSrmm API returns output data for LIST type of subcommands as structured fields when you specify OUTPUT=FIELDS. The Structured Field Introducers for each type of LIST subcommand are found in:

- “LISTBIN Structured Field Introducers”
- “LISTCONTROL Structured Field Introducers” on page 40
- “LISTDATASET Structured Field Introducers” on page 42
- “LISTOWNER Structured Field Introducers” on page 44
- “LISTPRODUCT Structured Field Introducers” on page 44
- “LISTRACK Structured Field Introducers” on page 44
- “LISTVOLUME Structured Field Introducers” on page 44
- “LISTVRS Structured Field Introducers” on page 47

LISTBIN Structured Field Introducers

The Structured Field Introducers produced for the LISTBIN subcommand with OUTPUT=FIELDS are shown in Figure 34 on page 40.

```
<Begin RACK or BIN group>
  <RCK  - rack or bin number       : 6  , character:         >
  <VOL  - volume serial            : 6  , character:         >
  <RST  - rack or bin status       : 1  , bin(8):            >
  <LOC  - location                 : 6  , character:         >
  <MEDN - media name               : 8  , character:         >
<End RACK or BIN Group>
```

Figure 34. SFIs for LISTBIN with OUTPUT=FIELDS

LISTCONTROL Structured Field Introducers

The Structured Field Introducers produced for the LISTCONTROL subcommand with OUTPUT=FIELDS are shown in Figure 35.

```
<Begin CONTROL group>
  <Begin CNTL group>
    <MTP  - CDS type                 : 1  , bin(8):            >
    <MDTJ - CDS create date          : 4  , packed decimal:    >
    <MTM  - CDS create time          : 4  , packed decimal:    >
    <JRNU - journal percentage used  : 2  , bin(15):           >
    <JRNF - JOURNALFULL parmlib value: 2  , bin(15):           >
    <BDTJ - last CDS backup date     : 4  , packed decimal:    >
    <BTM  - last CDS backup time     : 4  , packed decimal:    >
    <XDTJ - expiration date          : 4  , packed decimal:    >
    <XTM  - last inven mgmt exp time : 4  , packed decimal:    >
    <RDTJ - last CDS extract date    : 4  , packed decimal:    >
    <RTM  - last CDS extract time    : 4  , packed decimal:    >
    <DDTJ - last store update date   : 4  , packed decimal:    >
    <DTM  - last store update time   : 4  , packed decimal:    >
    <SOSJ - last XPROC start date    : 4  , packed decimal:    >
    <SOST - last XPROC start time    : 4  , packed decimal:    >
    <VDTJ - last VRSEL date          : 4  , packed decimal:    >
    <VTM  - last VRSEL time          : 4  , packed decimal:    >
    <LRK  - # LIBRARY rack numbers   : 4  , bin(31):           >
    <FRK  - free rack numbers in lib : 4  , bin(31):           >
    <LBN  - bin numbers in LOCAL     : 4  , bin(31):           >
    <FLB  - free bin numbers in LOCAL: 4  , bin(31):           >
    <DBN  - bin numbers in DISTANT   : 4  , bin(31):           >
    <FDB  - free bins in DISTANT loc : 4  , bin(31):           >
    <RBN  - # bin numbers in REMOTE  : 4  , bin(31):           >
    <FRB  - free bin nums in REMOTE  : 4  , bin(31):           >
    <CACT - control active functions : 1  , bit(8):            >
    <CSDT - Catalog Synchronize date : 4  , packed decimal:    >
    <CSTM - Catalog Synchronize time : 4  , packed decimal:    >
    <FCSP - Catalog Sync in progress : 1  , bin(8):            >
    <CSVE - Stacked volume enabled   : 1  , bin(8):            >
    <X100 - EDGUX100 exit status     : 1  , bin(8):            >
    <X200 - EDGUX200 exit status     : 1  , bin(8):            >
  <End CNTL group>
```

Figure 35. (Part 1 of 3) SFIs for LISTCONTROL with OUTPUT=FIELDS

```
  <Begin OPTION group>
    <OPM  - operating mode           : 1  , bin(8):           >
    <DRP  - default retention period : 4  , bin(31):          >
    <MRP  - maximum retention period : 4  , bin(31):          >
    <CRP  - CATRETPD retention period: 4  , bin(31):          >
    <MDS  - CDS data set name        : 44 , character:        >
    <JDS  - journal name             : 44 , character:        >
    <JRNF - JOURNALFULL parmlib value: 2  , bin(15):          >
    <CATS - CATSYSID value           : 1  , bin(8):           >
    <SOSP - scratch procedure name   : 8  , character:        >
    <BKPP - backup procedure name    : 8  , character:        >
    <IPL  - data check reqd in IPL   : 1  , bin(8):           >
    <DTE  - installation date format : 1  , bin(8):           >
    <RCF  - installation RACF support: 1  , bin(8):           >
    <AUD  - SMF audit record number  : 2  , bin(15):          >
    <SSM  - SMF security rcd number  : 2  , bin(15):          >
    <CDS  - control data set ID      : 8  , character:        >
    <SLM  - MAXHOLD value            : 2  , bin(15):          >
    <LCT  - default lines per page   : 2  , bin(15):          >
    <SID  - RMM system ID            : 8  , character:        >
    <BLP  - BLP option               : 1  , bin(8):           >
    <V1   - TLCS V1 option           : 1  , bin(8):           >
    <NOT  - Notify  option           : 1  , bin(8):           >
    <UNC  - uncatalog option         : 1  , bin(8):           >
    <VRJ  - VRS job name             : 1  , bin(8):           >
    <MSGF - case of message text     : 1  , bin(8):           >
    <MOP  - master overwrite         : 1  , bin(8):           >
    <ACCT - accounting source        : 1  , bin(8):           >
    <VCHG - VRSCHANGE value          : 1  , bin(8):           >
    <VRSL - VRSEL value              : 1  , bin(8):           >
    <PSFX - parmlib member suffix    : 2  , character:        >
    <VACT - VRSMIN action            : 1  , bin(8):           >
    <VMIN - VRSMIN count value       : 4  , bin(31):          >
    <DSPD - Disposition DD name      : 8  , character:        >
    <DSPM - Disposition message prefi: 8  , character:        >
    <RTBY - Retain by                : 1  , bin(8):           >
    <MVBY - Move by                  : 1  , bin(8):           >
    <PDA  - PDA state                : 1  , bin(8):           >
    <PDAC - PDA block count          : 1  , bin(8):           >
    <PDAS - PDA block size           : 1  , bin(8):           >
    <PDAL - PDA log state            : 1  , bin(8):           >
    <TVXP - Extradays retention      : 1  , bit(8):           >
    <SMP  - System-managed tape purge: 1  , bin(8):           >
    <SMU  - System-managed tape updat: 1  , bin(8):           >
<End OPTION group>
  <Begin SECCLS group>
    <SEC  - security class number    : 1  , bin(8):           >
    <NME  - security class name      : 8  , character:        >
    <SCST - sec class status         : 1  , bit(8):           >
    <CLS  - sec class description    : 32 , character:        >
  <End SECCLS group>
```

Figure 36. (Part 2 of 3) SFIs for LISTCONTROL with OUTPUT=FIELDS

```
    <Begin VLPOOL group>
    <PID  - pool prefix              : 6  , character:         >
    <PSN  - pool definition system ID: 8  , character:         >
    <PRF  - pool definition RACF opt : 1  , bin(8):            >
    <PTP  - pool definition pool type: 1  , bin(8):            >
    <XDC  - expiration date check    : 1  , bin(8):            >
    <PLN  - pool name                : 8  , character:         >
    <MEDN - media name               : 8  , character:         >
    <PDS  - pool description         : 40 , character:         >
  <End VLPOOL group>
  <Begin MNTMSG group>
    <MID  - mount message ID         : 12 , character:         >
    <SMI  - offset, message ID (msg) : 2  , bin(15):           >
    <OVL  - offset to volume serial  : 2  , bin(15):           >
    <OPL  - offset to rack num/poolid: 2  , bin(15):           >
  <End MNTMSG group>
  <Begin REJECT group>
    <GRK  - generic rack number      : 6  , character:         >
    <TAC  - reject prefix type       : 1  , bin(8):            >
  <End REJECT group>
  <Begin LOCDEF group>
    <LDDF - location definition exist: 1  , bin(8):            >
    <LDLC - location name            : 8  , character:         >
    <LDMT - location mgmt type       : 1  , bin(8):            >
    <LDLT - location type            : 1  , bin(8):            >
    <LDPR - location priority        : 4  , bin(31):           >
    <LDMN - location media name      : 8  , character:         >
  <End LOCDEF group>
  <Begin ACTIONS group>
    <ACT  - actions on release       : 1  , bit(8):            >
    <AST  - action status            : 1  , bit(8):            >
  <End ACTIONS group>
  <Begin MOVES group>
    <MFR  - source location name     : 8  , character:         >
    <MST  - move status              : 1  , bin(8):            >
    <MTO  - target location name     : 8  , character:         >
    <MTY  - move type                : 1  , bin(8):            >
  <End MOVES group>
<End CONTROL group>
```

Figure 37. (Part 3 of 3) SFIs for LISTCONTROL with OUTPUT=FIELDS

When there is no information for a subgroup, such as MOVES, for the LISTCONTROL subcommand, the DFSMSrmm API returns all of the SFIs in the subgroup with no data. For example, when there are no outstanding volume actions, the DFSMSrmm API returns the MOVES subgroup (MFR, MST, MTO and MTY) with not data.

LISTDATASET Structured Field Introducers

The Structured Field Introducers produced for the LISTDATASET subcommand with OUTPUT=FIELDS are shown in Figure 38 on page 43.

```
<Begin DATASET group>
  <DSN  - data set name            : 44 , character:         >
  <CJBN - job name                 : 8  , character:         >
  <VOL  - volume serial            : 6  , character:         >
  <OWN  - owner                    : 8  , character:         >
  <DSEQ - data set sequence        : 4  , bin(31):           >
  <DEV  - device number (address)  : 4  , hexadecimal:       >
  <FILE - physical file sequence   : 4  , bin(31):           >
  <CDTJ - create date              : 4  , packed decimal:    >
  <CTM  - create time              : 4  , packed decimal:    >
  <SYS  - SMF system id            : 8  , character:         >
  <BLKS - block size               : 4  , bin(31):           >
  <BLKC - block count              : 4  , bin(31):           >
  <LRCL - logical record length    : 4  , bin(31):           >
```

Figure 38. (Part 1 of 2) SFIs for LISTDATASET with OUTPUT=FIELDS

```
  <RCFM - record format            : 4  , character:         >
  <DC   - data class               : 8  , character:         >
  <DLWJ - date last written        : 4  , packed decimal:    >
  <DLRJ - date last read           : 4  , packed decimal:    >
  <STEP - step name                : 8  , character:         >
  <DD   - dd name                  : 8  , character:         >
  <MC   - management class         : 8  , character:         >
  <SG   - storage group name       : 8  , character:         >
  <SC   - storage class            : 8  , character:         >
  <VMV  - VRS management value     : 8  , character:         >
  <RTDJ - retention date           : 4  , packed decimal:    >
  <VTYP - Primary VRS type         : 1  , bin(8):            >
  <VJBN - Primary VRS jobn         : 8  , character:         >
  <VNME - Primary VRS name         : 44 , character:         >
  <VSCN - Primary VRS subchain name: 8  , character:         >
  <VSCD - Primary VRS subchain date: 4  , packed decimal:    >
  <VRSR - VRS retained             : 1  , bin(8):            >
  <NME  - security class name      : 8  , character:         >
  <CLS  - sec class description    : 32 , character:         >
  <ABND - Abend while open         : 1  , bin(8):            >
  <CTLG - Catalog status           : 1  , bin(8):            >
  <2JBN - Secondary VRS jobnme mask: 8  , character:         >
  <2NME - Secondary VRS mask       : 8  , character:         >
  <2SCN - Second. VRS subchain name: 8  , character:         >
  <2SCD - Second. VRS subchain date: 4  , packed decimal:    >
  <BLKT - Total block count        : 4  , bin(31):           >
  <CPGM - Creating program name    : 4  , bin(31):           >
  <LPGM - Last used program name   : 8  , character:         >
  <LJOB - Last used job            : 8  , character:         >
  <LSTP - Last used step name      : 4  , packed decimal:    >
  <LDD  - Last used DD name        : 4  , bin(31):           >
  <LDEV - Last drive               : 4  , bin(31):           >
  <DPCT - Percent of volume        : 4  , bin(31):           >
  <End DATASET group>
```

Figure 39. (Part 2 of 2) SFIs for LISTDATASET with OUTPUT=FIELDS

LISTOWNER Structured Field Introducers

The Structured Field Introducers produced for the LISTOWNER subcommand with OUTPUT=FIELDS are shown in Figure 40.

```
<Begin OWNER group>
  <OWN  - owner                    : 8  , character:         >
  <SUR  - owner's surname          : 20 , character:         >
  <FOR  - owner's forename         : 20 , character:         >
  <DPT  - owner's department       : 40 , character:         >
  <ADL  - address line             : 40 , character:         >
  <ADL  - address line             : 40 , character:         >
  <ADL  - address line             : 40 , character:         >
  <ITL  - owner's internal tel num : 8  , character:         >
  <ETL  - owner's external tele num: 20 , character:         >
  <EMU  - owner's user ID          : 8  , character:         >
  <EMN  - owner's node             : 8  , character:         >
  <VLN  - number of volumes        : 4  , bin(31):           >
<End OWNER group>
```

Figure 40. SFIs for LISTOWNER with OUTPUT=FIELDS

LISTPRODUCT Structured Field Introducers

The Structured Field Introducers produced for the LISTPRODUCT subcommand with OUTPUT=FIELDS are shown in Figure 41.

```
<Begin PRODUCT group>
  <PNUM - software product number  : 8  , character:         >
  <VER  - software product version : 6  , character:         >
  <OWN  - owner                    : 8  , character:         >
  <PNME - product software name    : 30 , character:         >
  <PDSC - product description      : 32 , character:         >
  <VOL  - volume serial            : 6  , character:         >
  <RCK  - rack or bin number       : 6  , character:         >
  <FCD  - product feature code     : 4  , character:         >
  <VLN  - number of volumes        : 4  , bin(31):           >
<End PRODUCT group>
```

Figure 41. SFIs for LISTPRODUCT with OUTPUT=FIELDS

LISTRACK Structured Field Introducers

The Structured Field Introducers produced for the LISTRACK subcommand with OUTPUT=FIELDS are shown in Figure 42.

```
<Begin RACK or BIN group>
  <RCK  - rack or bin number       : 6  , character:         >
  <VOL  - volume serial            : 6  , character:         >
  <RST  - rack or bin status       : 1  , bin(8):            >
  <LOC  - location                 : 8  , character:         >
  <MEDN - media name               : 8  , character:         >
<End RACK or BIN Group>
```

Figure 42. SFIs for LISTRACK with OUTPUT=FIELDS

LISTVOLUME Structured Field Introducers

The Structured Field Introducers produced for the LISTVOLUME subcommand with OUTPUT=FIELDS are shown in Figure 43 on page 45.

```
<Begin VOLUME group>
  <Begin VOL group>
    <VOL  - volume serial           : 6  , character:         >
    <RCK  - rack or bin number      : 6  , character:         >
    <OWN  - owner                   : 8  , character:         >
    <CJBN - job name                : 8  , character:         >
    <CDTJ - create date             : 4  , packed decimal:    >
```

Figure 43. (Part 1 of 3) SFIs for LISTVOLUME with OUTPUT=FIELDS

```
    <CTM  - create time             : 4  , packed decimal:    >
    <ADTJ - assigned date           : 4  , packed decimal:    >
    <ATM  - assigned time           : 4  , packed decimal:    >
    <XDTJ - expiration date         : 4  , packed decimal:    >
    <OXDJ - original expiration date : 4  , packed decimal:   >
    <RTDJ - retention date          : 4  , packed decimal:    >
    <DSN  - data set name           : 44 , character:         >
    <VST  - volume status           : 1  , bit(8):            >
    <OCE  - volume info recorded OCE : 1  , bin(8):           >
    <AVL  - volume availability     : 1  , bit(8):            >
    <LBL  - volume label type       : 1  , bit(8):            >
    <DEN  - media density           : 1  , bin(8):            >
    <MEDT - media type              : 1  , bin(8):            >
    <MEDR - media recording format  : 1  , bin(8):            >
    <MEDC - media compaction        : 1  , bin(8):            >
    <MEDA - media special attributes : 1  , bin(8):           >
    <ACT  - actions on release      : 1  , bit(8):            >
    <PEND - actions pending         : 1  , bit(8):            >
    <SG   - storage group name      : 8  , character:         >
    <LOAN - loan location           : 8  , character:         >
    <ACN  - account number          : 40 , character:         >
    <DESC - volume description      : 30 , character:         >
    <NME  - security class name     : 8  , character:         >
    <CLS  - sec class description   : 32 , character:         >
    <VRSI - Scratch Immediate       : 1  , bit(8):            >
    <VRXI - Expiration date ignore  : 1  , bit(8):            >
    <VOLT - Volume Type             : 1  , bit(8):            >
    <LVC  - Current label version   : 1  , bit(8):            >
    <LVN  - Requiredlabel version   : 1  , bit(8):            >
    <RBYS - Retain by set           : 1  , bit(8):            >
    <STVC - Stacked volume count    : 4  , bin(31):           >
  <End VOL group>
  <Begin ACCESS group>
    <OAC  - owner access            : 1  , bin(8):            >
    <VAC  - volume access           : 1  , bin(8):            >
    <LCID - last change user id     : 8  , character:         >
    <VM   - VM use                  : 1  , bin(8):            >
    <MVS  - MVS use                 : 1  , bin(8):            >
    <UID  - user id                 : 8  , character:         >
  <End ACCESS group>
```

Figure 44. (Part 2 of 3) SFIs for LISTVOLUME with OUTPUT=FIELDS

```
  <Begin STAT group>
    <DSC  - data set count           : 4  , bin(31):          >
    <DSR  - data set recording       : 1  , bin(8):           >
    <USEM - volume usage (kb)        : 4  , bin(31):          >
    <USEC - volume use count         : 4  , bin(31):          >
    <DLRJ - date last read           : 4  , packed decimal:   >
    <DLWJ - date last written        : 4  , packed decimal:   >
    <LDEV - last drive               : 4  , character:        >
    <SEQ  - volume sequence          : 4  , fixed(31):        >
    <MEDN - media name               : 8  , character:        >
    <PVL  - previous volume          : 6  , character:        >
    <NVL  - next volume              : 6  , character:        >
    <PNUM - software product number  : 8  , character:        >
    <VER  - software product version : 6  , character:        >
    <FCD  - product feature code     : 4  , character:        >
    <TRD  - temporary read errors    : 4  , bin(31):          >
    <TWT  - temporary write errors   : 4  , bin(31):          >
    <PRD  - permanent read errors    : 4  , bin(31):          >
    <PWT  - permanent write errors   : 4  , bin(31):          >
    <VCAP - volume capacity          : 4  , bin(31):          >
    <VPCT - volume percent full      : 1  , bin(8):           >
  <End STAT group>
  <Begin STORE group>
    <LOC  - location                 : 8  , character:        >
    <DEST - destination              : 8  , character:        >
    <INTR - volume intransit status  : 1  , bin(8):           >
    <LOCT - location type            : 1  , bin(8):           >
    <HLOC - home location            : 8  , character:        >
    <OLOC - old location             : 8  , character:        >
    <NLOC - required location        : 8  , character:        >
    <SDTJ - movement tracking date   : 4  , packed decimal:   >
    <MOVM - move mode                : 1  , bin(8):           >
    <BIN  - bin number               : 6  , character:        >
    <BMN  - bin number media name    : 8  , character:        >
    <OBN  - old bin number           : 6  , character:        >
    <OBMN - old bin number media name: 8  , character:        >
    <CTNR - Container                :16  , character:        >
  <End STORE group>
<End VOLUME group>
```

Figure 45. (Part 3 of 3) SFIs for LISTVOLUME with OUTPUT=FIELDS

LISTVRS Structured Field Introducers

The Structured Field Introducers produced for the LISTVRS subcommand with OUTPUT=FIELDS are shown in Figure 46.

```
<Begin VRS group>
  <VRS  - vital rcd specification  : 44 , character:         >
  <TYP  - VRS type                 : 1  , bit(8):            >
  <VJBN - Primary VRS job name     : 8  , character:         >
  <VRC  - vital record count       : 2  , bin(31):           >
  <RET  - retention type           : 3  , bin(8):            >
  <VDD  - VRS delay days           : 2  , bin(15):           >
  <LOC  - location                 : 8  , character:         >
  <SC1  - VRS store number         : 4  , bin(31):           >
  <PRTY - priority                 : 4  , bin(31):           >
  <NVRS - next VRS name            : 8  , character:         >
  <OWN  - owner                    : 8  , character:         >
  <DESC - volume or vrs description: 30 , character:         >
  <DDTJ - delete date              : 4  , packed decimal:    >
  <VANX - Next VRS Type            : 1  , bit(8):            >
  <VRSI - Scratch Immediate        : 1  , bit(8):            >
  <VRXI - Expiration date ignore   : 1  , bit(8):            >
<End VRS group>
```

Figure 46. SFIs for LISTVRS with OUTPUT=FIELDS

Search Type of Subcommands

The DFSMSrmm Search type of subcommands are: SEARCHBIN, SEARCHDATASET, SEARCHPRODUCT, SEARCHRACK, SEARCHVOLUME, and SEARCHVRS. You use these subcommands to obtain information from the DFSMSrmm control data set about resources defined to DFSMSrmm.

The DFSMSrmm API returns data for all SEARCH type of subcommands as structured fields when you specify OUTPUT=FIELDS.

The DFSMSrmm API returns expanded output data for the RMM TSO SEARCHDATASET, SEARCHPRODUCT, SEARCHVOLUME, and SEARCHVRS subcommands when you also specify the EXPAND=YES parameter.

SEARCHBIN Structured Field Introducers

Figure 47 shows the output that DFSMSrmm returns when you specify the SEARCHBIN subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=YES parameters.

```
<Begin RACK or BIN group>
  <RCK  - rack or bin number       : 6  , character:         >
  <VOL  - volume serial            : 6  , character:         >
  <RST  - rack or bin status       : 1  , bin(8):            >
  <LOC  - location                 : 6  , character:         >
  <MEDN - media name               : 8  , character:         >
<End RACK or BIN Group>
```

Figure 47. SFIs for SEARCHBIN with OUTPUT=FIELDS,EXPAND=YES

SEARCHDATASET Structured Field Introducers

Figure 48 on page 48 shows the output DFSMSrmm returns when you specify the SEARCHDATASET subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters.

The expanded output that DFSMSrmm returns when you specify the SEARCHDATASET subcommand with the OUTPUT=FIELDS and EXPAND=YES parameters is the same as shown in “LISTDATASET Structured Field Introducers” on page 42 for LISTDATASET.

```
<Begin DATASET group>
  <DSN  - data set name            : 44 , character:         >
  <VOL  - volume serial            : 6  , character:         >
  <OWN  - owner                    : 8  , character:         >
  <CDTJ - create date              : 4  , packed decimal:    >
  <CTM  - create time              : 4  , packed decimal:    >
  <FILE - physical file sequence   : 4  , bin(31):           >
<End DATASET group>
```

Figure 48. SFIs for SEARCHDATASET with OUTPUT=FIELDS,EXPAND=NO

SEARCHPRODUCT Structured Field Introducers

Figure 49 shows the output DFSMSrmm returns when you specify the SEARCHPRODUCT subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters.

The expanded output that DFSMSrmm returns when you specify the SEARCHPRODUCT subcommand with the OUTPUT=FIELDS and EXPAND=YES parameters is the same as shown in “LISTPRODUCT Structured Field Introducers” on page 44 for LISTPRODUCT.

```
<Begin PRODUCT group>
  <PNUM - software product number  : 8  , character:         >
  <VER  - software product version : 6  , character:         >
  <PNME - product software name    : 30 , character:         >
  <FCD  - product feature code     : 4  , character:         >
  <VLN  - number of volumes        : 4  , bin(31):           >
  <VOL  - volume serial            : 6  , character:         >
<End PRODUCT group>
```

Figure 49. SFIs for SEARCHPRODUCT with OUTPUT=FIELDS,EXPAND=NO

SEARCHRACK Structured Field Introducers

Figure 50 shows the output DFSMSrmm returns when you specify the SEARCHRACK subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters.

```
<Begin RACK or BIN group>
  <RCK  - rack or bin number       : 6  , character:         >
  <VOL  - volume serial            : 6  , character:         >
  <RST  - rack or bin status       : 1  , bin(8):            >
  <LOC  - location                 : 8  , character:         >
  <MEDN - media name               : 8  , character:         >
<End RACK or BIN Group>
```

Figure 50. SFIs for SEARCHRACK with OUTPUT=FIELDS,EXPAND=NO

SEARCHVOLUME Structured Field Introducers

Figure 51 on page 49 shows the output DFSMSrmm returns when you specify the SEARCHVOLUME subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters.

The expanded output that DFSMSrmm returns when you specify the SEARCHVOLUME subcommand with the OUTPUT=FIELDS and EXPAND=YES parameters is the same as shown in “LISTVOLUME Structured Field Introducers” on page 44 for LISTVOLUME.

```
<Begin VOLUME group>
  <VOL  - volume serial            : 6  , character:         >
  <OWN  - owner                    : 8  , character:         >
  <RCK  - rack or bin number       : 6  , character:         >
  <ADTJ - assigned date            : 4  , packed decimal:    >
  <XDTJ - expiration date          : 4  , packed decimal:    >
  <RTDJ - retention date           : 4  , packed decimal:    >
  <LOC  - location                 : 8  , character:         >
  <INTR - volume intransit status  : 1  , bin(8):            >
  <HLOC - home location            : 8  , character:         >
  <DSC  - data set count           : 4  , bin(31):           >
  <VST  - volume status            : 1  , bit(8):            >
  <AVL  - volume availability      : 1  , bit(8):            >
  <LBL  - volume label             : 1  , bit(8):            >
  <MEDT - media type               : 1  , bin(8):            >
  <MEDR - media recording format   : 1  , bin(8):            >
  <MEDC - media compaction         : 1  , bin(8):            >
  <MEDA - media special attributes : 1  , bin(8):            >
  <PEND - actions pending          : 1  , bit(8):            >
  <LOAN - loan location            : 8  , character:         >
  <DEST - destination              : 8  , character:         >
  <DSR  - data set recording       : 1  , bin(8)             >
  <SEQ  - volume sequence          : 4  , bin(31):           >
  <MEDN - media name               : 8  , character:         >
  <LVC  - Current label version    : 1  , bit(8):            >
  <LVN  - Required label version   : 1  , bit(8):            >
<End VOLUME group>
```

Figure 51. SFIs for SEARCHVOLUME with OUTPUT=FIELDS,EXPAND=NO

SEARCHVRS Structured Field Introducers

Figure 52 shows the output DFSMSrmm returns when you specify the SEARCHVRS subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters.

The expanded output that DFSMSrmm returns when you specify the SEARCHVRS subcommand with the OUTPUT=FIELDS and EXPAND=YES parameters is the same as shown in “LISTVRS Structured Field Introducers” on page 47 for LISTVRS.

```
<Begin VRS group>
  <VRS  - vital rcd specification  : 44 , character:         >
  <TYP  - VRS type                 : 1  , bit(8):            >
  <VJBN - Primary VRS job name     : 8  , character:         >
  <RET  - retention type           : 3  , bin(8):            >
  <LOC  - location                 : 8  , character:         >
  <PRTY - priority                 : 4  , bin(31):           >
  <NVRS - next VRS name            : 8  , character:         >
  <OWN  - owner                    : 8  , character:         >
  <DDTJ - delete date              : 4  , packed decimal:    >
  <VANX - Next VRS Type            : 1  , bit(8):            >
  <VRSI - Scratch Immediate        : 1  , bit(8):            >
  <VRXI - Expiration date ignore   : 1  , bit(8):            >
<End VRS group>
```

Figure 52. SFIs for SEARCHVRS with OUTPUT=FIELDS,EXPAND=NO

Controlling Output from List and Search Type Requests

The DFSMSrmm API returns information for a SEARCH type of subcommand or for a LISTCONTROL subcommand based on these factors:

- Whether you want line format or field format data.
- The size of your output buffer.
- The amount of output data.
- The LIMIT operand value used for a SEARCH type of subcommand.

Limiting the Search for a Request

Use the LIMIT keyword on SEARCH type of subcommands to limit the number of entries DFSMSrmm returns. To conserve use of system resources, such as dynamic storage, DFSMSrmm suspends a search operation after the number of entries matches the limit value you specify or the default limit value.

When you issue an RMM TSO Search type of subcommand, you can use the LIMIT operand to limit the number of entries returned. DFSMSrmm ends the search because the limit you set is reached or all available entries have been returned.

For an application program, the DFSMSrmm API causes DFSMSrmm to resume the search. LIMIT does not limit the total number of entries that the DFSMSrmm API returns to your application program and you cannot use LIMIT to end the subcommand before you have received all of the entries for a subcommand. Instead, you can specify OPERATION=CONTINUE regardless of whether limit has been reached, or begin a new command, or use EDGXCI OPERATION=RELEASE.

Output Buffer Examples

The examples in this section illustrate the following:

- SEARCH type subcommands (and LISTCONTROL) might require your application program to use one or more OPERATION=CONTINUE calls to the DFSMSrmm API to receive all of the search results.
- Your application program should expect to receive more than one set of return and reason codes. In the example, DFSMSrmm issued a different set of codes for each output buffer:
  - Return code 0, reason code 4.
  - Return code 4, reason code 2.
  - Return code 4, reason code 4.

  Depending on the subcommand you specify, the search criteria you specify (fully or partially qualified names), whether you specify a LIMIT value or LIMIT(*), DFSMSrmm might also issue the following:
  - Return code 0, reason code 0.
  - Return code 4, reason code 8.
- Header lines for search lists are placed at the beginning of the first output buffer of each set of buffers: The first output buffer after OPERATION=BEGIN, and the first output buffer after OPERATION=CONTINUE in response to the return code 4 and reason code 2.
- Messages issued by DFSMSrmm and that are placed in your output buffers are introduced by <MSGL> SFIs rather than <LINE> SFIs.
- The number of output data lines that are placed in your buffer is dependent upon the interaction of the following:

– The total number of searched records (entries).
– The size of your output buffer.
– The LIMIT value used for the search.

Figure 53, Figure 54 on page 52, and Figure 55 on page 52 display the contents of the output buffers when:

- Your application program issues an OPERATION=BEGIN, OUTPUT=LINES for a SEARCHRACK RACK(*) LIMIT(90) subcommand.
- Your application program is using a minimum size (4096 bytes) output buffer.
- There are 130 records in the RMM inventory.

First Output Buffer

The DFSMSrmm API issues return code 0 and reason code 4 and returns control to your application program. Your output buffer contains 78 structured fields.

In Figure 53:

- The group begins with the <Begin RACK or BIN group>.
- The structured fields between the Begin and End RACK group SFIs are all introduced by a <LINE> SFI.
- The first two lines after the Begin RACK group are the header lines for the list of RACK entries.
- The group ends with the <End RACK or BIN group>.

The DFSMSrmm API returns code 0 and reason code 4 which means there is more output data, so your application program should continue the subcommand request by using the EDGXCI macro OPERATION=CONTINUE parameter.

```
<Begin RACK or BIN group>
  <LINE>Rack    Medianame  Volume  Status    Location
  <LINE>------  ---------  ------  --------  --------
  <LINE>020610  CART3480   020610  IN USE    SHELF
  <LINE>020742  CART3480   020742  IN USE    SHELF
  <LINE>021042  CART3480   021042  IN USE    SHELF
    ...
    ...
  <LINE>030311  CART3480   030311  IN USE    SHELF
  <LINE>030318  CART3480   030318  IN USE    SHELF
<End RACK or BIN group>
```

Figure 53. CONTINUE Example, First Output Buffer

Second Output Buffer

After using the OPERATION=CONTINUE parameter, the DFSMSrmm API continues processing. The DFSMSrmm API issues return code 4 and reason code 2, returns control to your application program. Your output buffer contains 20 structured fields.

```
<Begin RACK or BIN group>
  <LINE>031086  CART3480   031086  IN USE    SHELF
  <LINE>031568  CART3480   031568  IN USE    SHELF
  <LINE>031599  CART3480   031599  IN USE    TRON
    ...
    ...
  <LINE>032848  CART3480   032848  IN USE    SHELF
  <LINE>032898  CART3480   032898  IN USE    SHELF
  <MSGL>EDG3203I SEARCH COMPLETE - MORE ENTRIES MAY EXIST
  <MSGL>EDG3012I 90         ENTRIES LISTED
<End RACK or BIN group>
```

Figure 54. CONTINUE Example, Second Output Buffer

In Figure 54:

- There are no header lines in the second output buffer.
- There are only 16 output data lines (the LINE SFIs).
- The last output data line is followed by two message lines introduced by the <MSGL> SFI.

The DFSMSrmm API returns control to your application program even though there is room in the output buffer for more data. This is because the LIMIT value of 90 was reached as indicated by the second message line.

The return code 4 and reason code 2 indicate that more entries might exist. When you use OPERATION=CONTINUE, one of the following is likely to occur:

- When there are more entries, your application program receives control back with more output data in your output buffer.
- When there are no other entries, your application program receives control back with a buffer that is empty or that contains only messages.

Third (Last) Output Buffer

After the second OPERATION=CONTINUE, control is returned to your application program with return code 4 and reason code 4, and your output buffer contains 45 structured fields.

```
<Begin RACK or BIN group>
  <LINE>Rack    Mediname  Volume  Status    Location
  <LINE>------  --------  ------  --------  --------
  <LINE>032935  CART3480   032935  IN USE    SHELF
  <LINE>032941  CART3480   032941  IN USE    SHELF
  <LINE>032946  CART3480   032946  IN USE    SHELF
    ...
    ...
  <LINE>070692  CART3480   070692  IN USE    SHELF
  <LINE>070693  CART3480   070693  IN USE    SHELF
  <MSGL>EDG3012I 40         ENTRIES LISTED
<End RACK or BIN group>
```

Figure 55. CONTINUE Example, Third (Last) Output Buffer

In Figure 55:

- The first two lines after the Begin RACK group are the header lines that you saw in the first output buffer. This is because the output is for a second search that the DFSMSrmm API started again when you specified OPERATION=CONTINUE in response to the return code 4 and reason code 2 that the DFSMSrmm API returned.

- The last output data line in your output buffer is followed by a single message line.
- The return code 4 and reason code 4 indicate that the subcommand was ended before the LIMIT value was reached.
- The total number of entries given to your application program in the three output buffers is 130: 74 in the first, 16 in the second, and 40 in the last output buffer.

Appendix A. Structured Field Introducers

This section defines the Structured Field Introducers used by the DFSMSrmm API to identify fields in API output.

SFI Format

All Structured Field Introducers (SFIs) have the following format:

Bytes	Description
0-1	2-byte length: SFI length plus data length
2-4	3-byte identifier: SFI ID (hexadecimal)
5	1-byte type modifier: Type of SFI • 0 = 8-byte, fixed-length SFI
6	1-byte (Reserved)
7	1-byte data type: Type of data, if any, that follows the SFI • 0=Undefined (no data) • 1=Character (fixed-length) • 2=Bit(8) (1-byte flag, multiple bits can be on) • 3=Binary(8) (1-byte (hex) value) • 4=Binary(15) (2-byte (hex) value) • 5=Binary(31) (4-byte (hex) value) • 6=Reserved • 7=Character (variable-length) • 8=Reserved • 9=(4 bytes) Packed decimal Julian date: yyyydddC • A=(4 bytes) Packed decimal time format: hhmmsstC

Structured Field Lengths

All structured fields have a minimum length of 8 bytes (for the Structured Field Introducer). The length can be fixed-length or variable-length.

- **Fixed-length**:

  The structured field has one of two length values: 8 when there is no data or the defined maximum length. For example, if the length is defined as X'000C' (decimal 12) for a particular structured field, the length in the SFI has a value of either X'0008' (no data) or X'000C' (data length = 4).

- **Variable-length**:

  The structured field can have a length that varies from 8 (no data) up to maximum stated size. For example, because a data set name varies from 1 to 44 characters in length, the length value in an SFI for a data set name can be X'0008' (no data), or it can vary from X'0009' to X'0034' (9 to 52 decimal).

SFIs for Begin and End Resource Groups

Begin and End Resource Group SFIs identify when the output for a particular resource begins and ends. Begin and End Resource groups can used to identify subgroups within a group. The Begin and End Resource groups are never followed by data. The SFIs that identify Begin and End resource groups are shown in Table 6 on page 56.

Table 6. Begin and End Group Structured Field Introducers

Begin - End IDs	Resource Group
X'021000' - X'021080'	ACCESS - within VOLUME
X'022000' - X'022080'	ACTIONS - within CONTROL
X'024000' - X'024080'	CNTL - within CONTROL
X'025000' - X'025080'	CONTROL
X'026000' - X'026080'	DATASET
X'027000' - X'027080'	LOCDEF - within CONTROL
X'028000' - X'028080'	MESSAGE
X'029000' - X'029080'	MNTMSG - within CONTROL
X'02A000' - X'02A080'	MOVES - within CONTROL
X'02B000' - X'02B080'	OPTION - within CONTROL
X'02C000' - X'02C080'	OWNER
X'02D000' - X'02D080'	PRODUCT
X'02E000' - X'02E080'	RACK or BIN
X'02F000' - X'02F080'	REJECT - within CONTROL
X'030000' - X'030080'	SECCLS - within CONTROL
X'031000' - X'031080'	SECLVL - within CONTROL
X'032000' - X'032080'	STAT - within VOLUME
X'033000' - X'033080'	STORE - within VOLUME
X'034000' - X'034080'	SYSRETC
X'035000' - X'035080'	VLPOOL - within CONTROL
X'036000' - X'036080'	VOL - within VOLUME
X'037000' - X'037080'	VOLUME
X'038000' - X'038080'	VRS

SFIs for Return and Reason Codes

The SFIs shown in Table 7 on page 57 introduce return and reason codes in your output buffer.

- The DFSMSrmm API issues the return and reason code SFIs only when the subcommand fails. Each return and reason code pair is grouped within the SYSRETC group. The FRC and FRS SFIs are used for return and reason codes returned from OAM. The RSNC and RTNC SFIs are used for return and reason codes from another system service.

  When the DFSMSrmm API builds a SYSRETC group for an error reported by a system service, look for additional information available from system messages in places like the terminal, SYSTSPRT, job log, and SYSLOG.
- Subcommands are described using standard DFSMSrmm abbreviations. For example, AV is for ADDVOLUME as shown in Table 3 on page 2.
- The SFI definitions are enclosed in single quotes (') to signify that they are 8-byte values, and two spaces are included in the IDs for readability.

Table 7. Reason and Return Code SFIs

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'400000'	FRC	12	Binary(31)	Function return code	AV CV DV GV
X'401000'	FRS	12	Binary(31)	Function reason code	AV CV DV GV
X'402000'	RSNC	12	Binary(31)	Reason code	Any subcommand
X'403000'	RTNC	12	Binary(31)	Return code	Any subcommand
X'404000'	SVCN	16	Character (variable length)	Service name	Any subcommand

SFIs for Messages and Message Variables

The SFIs described in Table 8 introduce messages and message variables that the DFSMSrmm API places in your output buffer:

- MSGL is used when OUTPUT=LINES.
- MSGN and ENTN are used when OUTPUT=FIELDS.
- The SFI definitions are enclosed in single quotes (') to signify that they are 8-byte values and the two spaces are inserted for readability.

The MSGN and ENTN SFIs are always grouped within the MESSAGE group. The MSGL SFIs are grouped within the MESSAGE group when the DFSMSrmm API is unable to determine which subcommand type the message is for. One or more SFIs other than ENTN might follow MSGN as described in “Messages and Message Variables SFIs” on page 36.

Table 8. Message SFIs

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'051000'	MSGL	259	Character (variable length)	Message line	Any subcommand
X'052000'	MSGN	16	Character (fixed length)	Message number ID	As previously defined
X'053000'	ENTN	12	Binary(31)	Number of entries Min 0, Max 10-digit	As previously defined
X'054000'	KEYF	64	Character (variable length)	Key from	SD SV
X'054200'	KEYT	64	Character (variable length)	Key to	SD SV
X'055000'	TYPF	16	Character (variable length)	VOLUME or DATASET	SD SV
X'055200'	TYPT	16	Character (variable length)	VOLUME or DATASET	SD SV

SFIs for Subcommand Output Data

The SFIs described in Table 9 introduce subcommand output data in your output buffer. These SFIs are always grouped within a pair of Begin and End Resource group SFIs.

The following notation is used:

- Subcommands are described using standard DFSMSrmm abbreviations. For example, LV is for LISTVOLUME and SS is for SEARCHVRS as described in Table 3 on page 2.
- The (e) following a search type of subcommand abbreviation means the expanded output is available if you specify EXPAND=YES. The absence of (e) means the SFI is used for both EXPAND=NO and EXPAND=YES.
- The range of two-byte and four-byte numbers is denoted by the minimum expected value and the maximum number of digits the number is expected to have. For example: “Min 1, Max 4-digit” means the minimum expected value of the number is one and the maximum expected number of digits in the number is four.
- The SFI definitions are enclosed in single quotes (’) to signify that they are 8-byte values and the two spaces are inserted for readability. Bit data (flags) values are also enclosed in single quotes.

Table 9. Command SFIs

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'800500'	ABND	9	Binary(8)	Abend while open 0=NO 1=YES	LD SD(e)
X'800800'	ACCT	9	Binary(8)	Accounting source 0=JOB 1=STEP	LC
X'801000'	ACN	48	Character (variable length)	Account number	LV SV(e)
X'802000'	ACT	9	Bit(8)	Actions on release ’80’=SCRATCH ’40’=REPLACE ’20’=INIT ’10’=ERASE ’08’=RETURN ’04’=NOTIFY	LC LV SV(e)
X'803001'	ADL	48	Character (variable length)	Address line ID is incremented by one for each SFI	LO
X'804000'	ADTJ	12	Packed decimal Julian date format	Assigned date	LV SV
X'805000'	AST	9	Bit(8)	Action status ’80’=PENDING ’40’=CONFIRMED ’20’=COMPLETE ’10’=UNKNOWN	LC
X'806000'	ATM	12	Packed decimal time format	Assigned time	LV SV(e)
X'807000'	AUD	10	Binary(8)	SMF audit record number: 128-155	LC

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'808000'	AVL	9	Bit(8)	Volume availability ’40’=PENDREL ’20’=VITALRCD ’08’=ONLOAN ’04’=OPEN	LV SV
X'809000'	BDTJ	12	Packed decimal Julian date format	Last control data set backup date	LC
X'80A000'	BIN	14	Character (fixed length)	6-character alphameric bin number	LV SV(e)
X'80B000'	BKPP	16	Character (Variable length)	Backup procedure name	LC
X'80C000'	BLKC	12	Binary(31)	Block count: Min 0, Max 5-digit	LD SD(e)
X'80D000'	BLKS	12	Binary(31)	Block size: Min 0, Max 5-digit	LD SD(e)
X'80D030'	BLKT	12	Binary(31)	Total block count	LD SD(e)
X'80E000'	BLP	9	Binary(8)	BLP option: 0=RMM 1=NORMM	LC
X'80F000'	BMN	16	Character (variable length)	Bin number media name	LV SV(e)
X'810000'	BTM	12	Packed decimal time format	Last control data set backup time	LC
X'811000'	CACT	9	Bit(8)	Control active functions ’80’=BACKUP ’40’=RESTORE ’20’=VERIFY ’10’=EXPROC ’08’=EXTRACT ’04’=DSTORE ’02’=VRSEL	LC
X'811800'	CATS	9	Binary(8)	CATSYSID value 0=SET 1=NOTSET 2=*	LC
X'812000'	CDS	16	Character (variable length)	Control data set identifier	LC
X'813000'	CDTJ	12	Packed decimal Julian date format	Create date	LD LV SD SV(e)
X'814000'	CJBN	16	Character (variable length)	Job name	LD LV SD(e) SV(e)
X'815000'	CLIB	16	Character (variable length)	Current library name	AV CV DV
X'816000'	CLS	40	Character (variable length)	Security class description	LC LD LV SD(e) SV(e)
X'817000'	CNT	12	Binary(31)	Bin, rack, or volume count: Min 0, Max 5-digit	AB AR AV DB DR

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'817820'	CPGM	16	Character (fixed length)	Creating program name	LD SD(e)
X'818000'	CRP	12	Binary(31)	CATRETPD retention period: Min 0, Max 4-digit	LC
X'818800'	CSDT	12	Packed decimal Julian date	Catalog synchronize date	LC
X'819000'	CSG	16	Character (variable length)	Current storage group name	AV CV
X'819400'	CSTM	12	Packed decimal time date	Catalog synchronize time	LC
X'819600'	CSVE	9	Binary(8)	Stacked volume enable status: 0=None 1=Enabled 2=Disabled 3=Mixed	LC
X'819800'	CTLG	9	Binary(8)	Catalog status: 0=UNKNOWN 1=NO 2=YES	LD SD(e)
X'81A000'	CTM	12	Packed decimal time format	Create time	LD LV SD SV(e)
X'81A300'	CTNR	24	Character (variable length)	In container	LV STORE
X'81B000'	DBN	12	Binary(31)	Bin numbers in DISTANT location: Min 0, Max 6-digit	LC
X'81C000'	DC	16	Character (variable length)	Data class name	LD SD(e)
X'81D000'	DD	16	Character (variable length)	DD name	LD SD(e)
X'81E000'	DDTJ	12	Packed decimal Julian date format	Delete date or last store update date	LC LS SS
X'81F000'	DEN	9	Binary(8)	Media density: 0=UNDEFINED 1=1600 2=6250 3=3480 4=COMPACT	LV SV(e)
X'820000'	DESC	38	Character (variable length)	Volume or VRS description	LS LV SS(e) SV(e)
X'821000'	DEST	16	Character (variable length)	Destination name	LV SV
X'822000'	DEV	12	Character (fixed length)	Device number	LD SD(e)
X'823000'	DLRJ	12	Packed decimal Julian date format	Date last read	LD LV SD(e) SV(e)
X'824000'	DLWJ	12	Packed decimal Julian date format	Date last written	LD LV SD(e) SV(e)

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'825000'	DNM	52	Character (variable length)	Data set name mask	LC
X'825E00'	DPCT	9	Binary(8)	Percent of volume	LD SD(e)
X'826000'	DPT	48	Character (variable length)	Owner's department	LO
X'827000'	DRP	12	Binary(31)	Default retention period: Min 0, Max 4-digit	LC
X'828000'	DSC	12	Binary(31)	Data set count: Min 0, Max 4-digit	LV SV
X'829000'	DSEQ	12	Binary(31)	Data set sequence: Min 0, Max 4-digit	LD SD(e)
X'82A000'	DSN	52	Character (variable length)	Data set name	LD LV SD SV(e)
X'82A500'	DSPD	16	Character (variable length)	Disposition DD name	LC
X'82AA00'	DSPM	16	Character (variable length)	Disposition message prefix	LC
X'82B000'	DSR	9	Binary(8)	Data set recording: 0=OFF 1=ON	LV SV
X'82C000'	DTE	9	Binary(8)	Installation date format: 1=A 2=E 3=I 4=J	LC
X'82D000'	DTM	12	Packed decimal time format	Last store update run time	LC
X'82E000'	EMN	16	Character (variable length)	Owner's node	LO
X'82F000'	EMU	16	Character (variable length)	Owner's user ID	LO
X'830000'	ETL	28	Character (variable length)	Telephone number	LO
X'831000'	FCD	12	Character (variable length)	Feature code	LP LV SP SV(e)
X'831800'	FCSP	9	Binary(8)	Catalog synchronize in progress: 0=NO 1=YES	LC
X'832000'	FDB	12	Binary(31)	Free bins in DISTANT location Min 0, Max 6-digit	LC
X'833000'	FILE	12	Binary(31)	Physical file sequence Min 1, Max 4-digit	LD SD
X'834000'	FLB	12	Binary(31)	Free bin numbers in LOCAL location: Min 0, Max 6-digit	LC

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'835000'	FOR	28	Character (variable length)	Owner's forename	LO
X'836000'	FRB	12	Binary(31)	Free bin numbers in REMOTE location: Min 0, Max 6-digit	LC
X'837000'	FRK	12	Binary(31)	Free rack numbers in library: Min 0, Max 10-digit	LC
X'838000'	GRK	14	Character (fixed length)	Generic rack number = reject prefix	LC
X'839000'	HLOC	16	Character (variable length)	Home location	LV SV
X'83A000'	INTR	9	Binary(8)	Volume intransit status: 0=NO 1=YES	LV SV
X'83B000'	IPL	9	Binary(8)	Data check required in IPL: 0=NO 1=YES	LC
X'83C000'	ITL	16	Character (variable length)	Telephone number	LO
X'83D000'	JDS	52	Character (variable length)	Journal name	LC
X'83E000'	JRNF	10	Binary(15)	JOURNALFULL parmlib value: 0 - 99	LC
X'83F000'	JRNU	10	Binary(15)	Journal percentage used: 0 - 100	LC
X'840000'	LBL	9	Bit(8)	Volume label type: '20'=NL '10'=AL '08'=SL '02'=BLP '01'=UL	LV SV
X'841000'	LBN	12	Binary(31)	Bin numbers in LOCAL location Min 0, Max 6-digit	LC
X'842000'	LCID	16	Last change user ID	LV SV(e)	Character (variable length)
X'843000'	LCT	10	Binary(15)	Default lines per page Min 10, Max 3-digit	LC
X'844000'	LDDF	9	Binary(8)	Location definition exists: 0=NO 1=YES	LC
X'843B00'	LDD	16	Character (fixed length)	Last used DD name	LD SD(e)
X'845000'	LDEV	16	Character (fixed length)	Last drive	LD SD(e) LV SV(e)

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'846000'	LDLC	16	Character (variable length)	Location name	LC
X'847000'	LDLT	9	Binary(8)	Location type: 0=SHELF 1=AUTO 2=MANUAL 3=STORE	LC
X'848000'	LDMN	16	Character (variable length)	Location media name	LC
X'849000'	LDMT	9	Binary(8)	Location management type: 0=UNDEFINED 1=BIN 2=NOBINS	LC
X'84A000'	LDPR	12	Binary(31)	Location priority: Min 0, Max 4-digit	LC
X'84B000'	LINE	264	Character (variable length)	Output data line	All list and search subcommands
X'84B420'	LJOB	16	Character (fixed length)	Last used job name	LD SD(e)
X'84C000'	LOAN	16	Character (fixed length)	Loan location	LV SV
X'84D000'	LOC	16	Character (fixed length)	Location	LB LR LS LV SB SR SS SV
X'84E000'	LOCT	9	Binary(8)	Location type 0=SHELF 1=STORE_BUILTIN_BINS 2=MANUAL 3=AUTO 4=STORE_BINS 5=STORE_NOBINS 6=IN CONTAINER	LV SV(e)
X'84E760'	LPGM	16	Character (fixed length)	Last used program name	LD SD(e)
X'84F000'	LRCL	12	Binary(31)	Logical record length: Min 0, Max 5-digit	LD SD(e)
X'850000'	LRK	12	Binary(31)	Library rack numbers: Min 0, Max 10-digit	LC
X'850370'	LSTP	16	Character (variable length)	Last used step name	LD SD(e)
X'850500'	LVC	9	Binary(8)	Current label version: 0=No version specified 1=Label version 1 specified 3=Label version 3 specified 4=Label version 4 specified	LV SV

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'850A00'	LVN	9	Binary(8)	Required label version: 0=No version specified 3=Label version 3 specified 4=Label version 4 specified	LV SV
X'851000'	MC	16	Character (variable length)	Management class	LD SD(e)
X'852000'	MDS	52	Character (variable length)	Control data set name	LC
X'853000'	MDTJ	12	Packed decimal Julian date format	Control data set create date	LC
X'854000'	MEDA	9	Binary(8)	Media special attributes: 0=NONE 1=READCOMP	LV SV
X'855000'	MEDC	9	Binary(8)	Media compaction 0=UNDEFINED 1=NO 2=YES	LV SV
X'856000'	MEDN	16	Character (variable length)	Media name	CV LC LB LR LV SB SR SV
X'857000'	MEDR	9	Binary(8)	Recording technology: 0=Non-cartridge 1=18TRK 2=36TRK 3=128TRK 4=256TRK	LV SV
X'858000'	MEDT	9	Binary(8)	Media type: 0=UNDEFINED 1=CST 2=ECCST 3=HPCT 4=EHPCT	LV SV
X'859000'	MFR	16	Character (variable length)	Source location name	LC
X'85A000'	MID	20	Character (variable length)	Mount message ID	LC
X'85B000'	MOVM	9	Binary(8)	Move mode: 0=AUTO 1=MANUAL	LV SV(e)
X'85C000'	MOP	9	Binary(8)	Master overwrite: 1=ADD 2=LAST 3=MATCH 4=USER	LC
X'85D000'	MRP	12	Binary(31)	Maximum retention period: Min 0, Max 4-digit -1 (negative) means unlimited retention.	LC
X'85E000'	MSGF	9	Binary(8)	Message text case: 0=MIXED 1=UPPER	LC

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'85F000'	MST	9	Binary(8)	Move status: 0=UNKNOWN 1=PENDING 2=CONFIRMED 3=COMPLETE	LC
X'860000'	MTM	12	Packed decimal time format	Control data set create time	LC
X'861000'	MTO	16	Character (variable length)	Target location name, installation defined name, SHELF, or SMS library name	LC
X'862000'	MTP	9	Binary(8)	Control data set type: 0=MASTER	LC
X'862800'	MTY	9	Binary(8)	Move type: 0=NOTRTS 1=RTS	LC
X'862B00'	MVBY	9	Binary(8)	Move by: 0=VOLUME 1=SET	LC
X'863000'	MVS	9	Binary(8)	MVS use 0=NO 1=YES	LV SV(e)
X'865000'	NLOC	16	Character (variable length)	Required location	LV SV(e)
X'866000'	NME	16	Character (variable length)	Security class name	LC LD LV SD(e) SV(e)
X'866800'	NOT	9	Binary(8)	User notification: 0=NO 1=YES	LC
X'867000'	NVL	14	Character (fixed length)	Next volume serial	LV SV(e)
X'868000'	NVRS	16	Character (variable length)	Next VRS name	LS SS
X'869000'	OAC	9	Binary(8)	Owner access 0=READ 1=UPDATE 2=ALTER	LV SV(e)
X'86A000'	OBMN	16	Character (variable length)	Old bin number media name	LV SV(e)
X'86B000'	OBN	14	Character (fixed length)	Old bin number	LV SV(e)
X'86B800'	OCE	9	Binary(8)	Volume information recorded at O/C/EOV 0=NO 1=YES	LV SV(e)
X'86C000'	OLOC	16	Character (variable length)	Old location	LV SV(e)
X'86D000'	OPL	10	Binary(15)	Position of rack number or pool ID Min 1, Max 3-digit	LC Position in the message.

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'86E000'	OPM	9	Binary(8)	Operating mode 1=M 2=R 3=W 4=P	LC
X'86F000'	OVL	10	Binary(15)	Position of volume serial number: Min 1, Max 3-digit	LC Position in the message.
X'870000'	OWN	16	Character (variable length)	Owner	GV LD LO LP LS LV SD(e) SP SS SV
X'871000'	OXDJ	12	Packed decimal Julian date format	Original expiration date	LV SV(e)
X'871E00'	PDA	9	Binary(8)	PDA state: 0=Off 1=On 2=None	LC
X'871E10'	PDAC	9	Binary(8)	PDA block count: Numeric 2-255	LC
X'871E30'	PDAL	9	Binary(8)	PDA log state: 0=Off 1=On	LC
X'871E90'	PDAS	9	Binary(8)	PDA block size: Numeric 1-31	LC
X'872000'	PDS	48	Character (variable length)	Pool description	LC
X'873000'	PDSC	40	Character (variable length)	Product description	LP SP(e)
X'874000'	PEND	9	Bit(8)	Actions pending: '80'=SCRATCH '40'=REPLACE '20'=INIT '10'=ERASE '08'=RETURN '04'=NOTIFY	LV SV
X'875000'	PID	14	Character (variable length)	Pool prefix	LC
X'876000'	PLN	16	Character (variable length)	Pool name	LC
X'877000'	PNME	38	Character (variable length)	Software product name	LP SP
X'878000'	PNUM	16	Character (variable length)	Software product number	LP LV SP SV(e)
X'879000'	PRD	12	Binary(31)	Permanent read errors: Min 0, Max 5-digit	LV SV(e)

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'87A000'	PRF	9	Binary(8)	Pool definition RACF® (A component of the SecureWay® Security Server for OS/390) option: 0=NO 1=YES	LC
X'87B000'	PRTY	12	Binary(31)	Priority: Min 0, Max 4-digit	LS SS
X'87C000'	PSFX	10	Character (fixed length)	Parmlib member suffix	LC
X'87D000'	PSN	16	Character (variable length)	Pool definition system ID	LC
X'87E000'	PTP	9	Binary(8)	Pool definition pool type: 0=SCRATCH 1=RACK	LC
X'87F000'	PVL	14	Character (fixed length)	Previous volume: 1 - 6 character	LV SV(e)
X'880000'	PWT	12	Binary(31)	Permanent write errors: Min 0, Max 5-digit	LV SV(e)
X'881000'	RBN	12	Binary(31)	Number of bin numbers in REMOTE location: Min 0, Max 6-digit	LC
X'881200'	RBYS	9	Binary(8)	Retain by set: 0=No 1=Yes	LV SV(e)
X'882000'	RCF	9	Binary(8)	Installation RACF support: 1=N 2=P 3=A	LC
X'883000'	RCFM	12	Character (variable length)	RECFM	LD SD(e)
X'884000'	RCK	14	Character (fixed length)	Rack or bin number	AB AR AV CV DB DR LB LP LR LV SB SP(e) SR SV
X'886000'	RDTJ	12	Packed decimal Julian date format	Last control data set extract date	LC
X'888000'	RET	11	Binary(8)	Retention type 1st byte: 1=RETAIN_WHILE_CATALOGED 2nd byte: 1=RETAIN_UNTIL_EXPIRED 3rd byte: 1=CYCLES 2=DAYS 3=REFDAYS 4=VOLUMES 5=EXTRA DAYS 6=BY DAYS CYCLE	LS SS

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'88A000'	RST	9	Binary(8)	Rack or bin status 0=EMPTY 1=FREE 2=INUSE	LB LR SB SR
X'88B900'	RTBY	9	Binary(8)	Retain by: 0=VOLUME 1=SET	LC
X'88C000'	RTDJ	12	Packed decimal Julian date format	Retention date	LD LV SD(e) SV
X'88E000'	RTM	12	Packed decimal time format	Last control data set extract time	LC
X'890000'	SC	16	Character (variable length)	Storage class name	LD SD(e)
X'892000'	SCST	9	Bit(8)	Security class status '80'=SMF '40'=MSGOPT '20'=ERASE	LC
X'894000'	SC1	12	Binary(31)	Storenumber Min 1, Max 5-digit	LS SS(e)
X'895000'	SDTJ	12	Packed decimal Julian date format	Movement tracking date	LV SV(e)
X'896000'	SEC	9	Binary(8)	Security class number Min 0, Max 255	LC
X'898000'	SEQ	12	Binary(31)	Volume sequence Min 1, Max 4-digit	LV
X'89A000'	SG	16	Character (variable length)	Storage group name	LD LV SD(e) SV(e)
X'89B000'	SID	16	Character (variable length)	DFSMSrmm system ID	LC
X'89C000'	SLM	10	Binary(15)	MAXHOLD value Min 10, Max 500	LC
X'89E000'	SMI'	10	Binary(15)	Offset to message ID Min 0, Max 3-digit	LC
X'89E210'	SMP	9	Binary(8)	System-managed tape purge: 0=NO 1=YES 2=ASIS	LC
X'89E220'	SMU	9	Bit(8)	System-managed tape update: 20=Command 40=Scratch 80=Exits N/A	LC
X'89F000'	SOSJ	12	Packed decimal Julian date format	Last expiration processing start date	LC
X'8A0000'	SOSP	16	Character (variable length)	Scratch procedure name	LC

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'8A1000'	SOST	12	Packed decimal time format	Last expiration processing start time	LC
X'8A2000'	SSM	10	Binary(15)	SMF security record number 128-155 = record number	LC
X'8A3000'	STEP	16	Character (variable length)	Step name	LD SD(e)
X'8A3800'	STVC	12	Binary(31)	Count of volumes stacked on a stacked volume	LV VOL SV(e)
X'8A4000'	SUR	28	Character (variable length)	Surname	LO
X'8A5000'	SYS	16	Character (variable length)	SMF System ID	LD SD(e)
X'8A6000'	TAC	9	Binary(8)	Reject type 0=ANYUSE 1=OUTPUT	LC
X'8A7000'	TRD	12	Binary(31)	Temporary read errors Min 0, Max 5-digit	LV SV(e)
X'8A7900'	TVXP	9	Binary(8)	Extradays retention: 0=RELEASE 1=EXPIRE 2=NONE	LC
X'8A8000'	TWT	12	Binary(31)	Temporary write errors: Min 0, Max 5-digit	LV SV(e)
X'8A9000'	TYP	9	Bit(8)	VRS type: '80'=GDG '40'=PSEUDGDG '20'=DSNAME '10'=VOLUME '08'=NAME	LS SS(e)
X'8AB001'	UID	16	Character (variable length)	User ID ID is incremented by one for each SFI	LV SV(e)
X'8AC000'	UNC	9	Binary(8)	Uncatalog option: 0=N 1=Y 2=S	LC
X'8AD000'	USEC	12	Binary(31)	Volume use count: Min 0, Max 5-digit	LV SV(e)
X'8AE000'	USEM	12	Binary(31)	Volume usage (KB): Min 0, Max 10-digit	LV SV(e)
X'8AF001'	VAC	9	Binary(8)	Volume access: 0=NONE 1=READ 2=UPDATE	LV SV(e)
X'8B0000'	VACT	9	Binary(8)	VRSMIN action: 0=FAIL 1=INFO 2=WARN	LC

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'8B0800'	VANX	9’	Binary(8)	Next VRS type: 0=Undefined1=Next2=And	LS SS
X'8B0B00'	VCAP	12	Binary(31)	Volume capacity	LV SV(e)
X'8B1000'	VCHG	9	Binary(8)	VRSCHANGE value: 0=INFO 1=VERIFY	LC
X'8B2000'	VDD	10	Binary(15)	VRS delay days: Min 0, Max 99	LS SS(e)
X'8B3000'	VDTJ	12	Packed decimal time format	Last inventory management processing date	LC
X'8B4000'	VER	14	Character (variable length)	Software produce version, release, modification **vvrrmm**	LP LV SP SV(e)
X'8B5000'	VJBN	16	Character (variable length)	Primary VRS job name	LD LS SD(e) SS
X'8B6000'	VLN	12	Binary(31)	Number of volumes: Min 0, Max 3-digit	LO LP SP
X'8B7000'	VM	9	Binary(31)	VM use: 0=NO 1=YES	LV SV(e)
X'8B8000'	VMIN	12	Binary(31)	VRSMIN count value: Min 0, Max 6-digit	LC
X'8B9000'	VMV	16	Character (variable length)	VRS management value	LD SD(e)
X'8BA000'	VNME	52	Character (variable length)	Primary VRS name	LD SD(e)
X'8BC000'	VOL	14	Character (fixed length)	1-6 characters volume serial (or $, @ or #)	AV CV GV LB LD LP LR LS LV SB SD SP SR SS SV
X'8BC200'	VOLT	9	Binary(8)	Volume type: 0=physical volume 1=logical volume	LV VOL
X'8BC300'	VPCT	9	Binary(8)	Volume percent full	LV SV(e)
X'8BD000'	VRC	12	Binary(31)	Vital record count: Min 1, Max 5-digit	LS SS(e)
X'8BE000'	VRJ	9	Binary(8)	VRS job name: 1 or 2	LC
X'8BF000'	VRS	52	Character (variable length)	Vital record specification name	LS SS
X'8BF500'	VRSI	9	Binary(8)	Release action scratch immediate: 0=NO 1=YES	LS LV SS SV(e)
X'8BFA00'	VRSL	9	Binary(8)	VRSEL value: 0=OLD, 1=NEW	LC

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'8C0000'	VRSR	9	Binary(8)	VRS retained status: 0=NO 1=YES	LD SD(e)
X'8C0800'	VRXI	9	Binary(8)	Expiration date ignore: 0=NO 1=YES	LV LS SS SV(e)
X'8C1000'	VSCD	12	Packed decimal Julian date format	Primary VRS subchain start date	LD SD(e)
X'8C1800'	VSCN	16	Character (variable length)	Primary VRS subchain name	LD SD(e)
X'8C2000'	VST	9	Bit(8)	Volume status: '80'=MASTER '40'=SCRATCH '20'=USER '10'=INIT '08'=ENTRY	LV SV
X'8C3000'	VTM	12	Packed decimal time format	Last inventory management VRS time	LC
X'8C4000'	VTYP	9	Binary(8)	Matching VRS type: 0=UNDEFINED 1=DATASET 2=SMSMC 3=VRSMV 4=DSNMV 5=DSNMC	LD SD(e)
X'8C4800'	V1	9	Binary(8)	TLCS V1: 0=NO 1=YES	LC
X'8C5000'	XDC	9	Binary(8)	Expiration date check: 0=NO 1=YES 2=OPERATOR	LC
X'8C6000'	XDTJ	12	Packed decimal Julian date format	Expiration date	LC LV SV
X'8C7000'	XTM	12	Packed decimal time format	Last inventory management expiration time	LC
X'8C7800'	X100	9	Binary(8)	EDGUX100 installation exit status: 0=No exit 1=Enabled 2=Disabled	LC
X'8C7801'	X200	9	Binary(8)	EDGUX200 installation exit status: 0=No exit 1=Enabled 2=Disabled	LC
X'8C8000'	2JBN	16	Character (variable length)	Secondary VRS jobname mask	LD SD(e)

Table 9. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'8C9000'	2NME	16	Character (variable length)	Secondary VRS mask	LD SD(e)
X'8CA000'	2SCD	12	Packed decimal Julian date format	Secondary VRS subchain start date	LD SD(e)
X'8CB000'	2SCN	16	Character (variable length)	Secondary VRS subchain name	LD SD(e)

Appendix B. Structured Field Introducers by Subcommand

Table 10 lists the structured field introducers by DFSMSrmm TSO subcommand.

Note: The RMM SEARCHDATASET, RMM SEARCHPRODUCT, RMM SEARCHVOLUME, and RMM SEARCHVRS subcommands return different sets of SFIs depending on if you specify the EDGXCI macro EXPAND=YES or EXPAND=NO parameter. When you specify the EXPAND=YES parameter, these subcommands return the same information as their corresponding RMM LIST subcommands, RMM LISTDATASET, RMM LISTPRODUCT, RMM LISTVOLUME, and RMM LISTVRS.

Table 10. Structured Field Introducers by Subcommand

Subcommand	Structured Field Introducers
ADDBIN	CNT ENTN MSGL MSGN RCK RSNC RTNC SVCN
ADDDATASET	ENTN MSGL MSGN RSNC RTNC SVCN
ADDOWNER	ENTN MSGL MSGN RSNC RTNC SVCN
ADDPRODUCT	ENTN MSGL MSGN RSNC RTNC SVCN
ADDRACK	CNT ENTN MSGL MSGN RCK RSNC RTNC SVCN
ADDVOLUME	CLIB CNT CSG ENTN FRC FRS MSGL MSGN RCK RSNC RTNC SVCN VOL
ADDVRS	ENTN MSGL MSGN RSNC RTNC SVCN
CHANGEDATASET	ENTN MSGL MSGN RSNC RTNC SVCN
CHANGEOWNER	ENTN MSGL MSGN RSNC RTNC SVCN
CHANGEPRODUCT	ENTN MSGL MSGN RSNC RTNC SVCN
CHANGEVOLUME	CLIB CSG ENTN FRC FRS MEDN MSGL MSGN RCK RSNC RTNC SVCN
DELETEBIN	CNT ENTN MSGL MSGN RCK RSNC RTNC SVCN
DELETEDATASET	ENTN MSGL MSGN RSNC RTNC SVCN
DELETEOWNER	ENTN MSGL MSGN RSNC RTNC SVCN
DELETEPRODUCT	ENTN MSGL MSGN RSNC RTNC SVCN
DELETERACK	CNT ENTN MSGL MSGN RCK RSNC RTNC SVCN
DELETEVOLUME	CLIB ENTN FRC FRS MSGL MSGN RSNC RTNC SVCN
DELETEVRS	ENTN MSGL MSGN RSNC RTNC SVCN
GETVOLUME	ENTN FRC FRS MSGL MSGN OWN RSNC RTNC SVCN VOL
LISTBIN	ENTN LINE LOC MEDN MSGL MSGN RCK RSNC RST RTNC SVCN VOL
LISTCONTROL	ACCT ACT AST AUD BDTJ BKPP BLP BTM CACT CATS CDS CLS CRP CSDT CSTM CSVE DBN DDTJ DNM DRP DSPD DSPM DTE DTM ENTN FCSP FDB FLB FRB FRK GRK IPL JDS JRNF JRNU LBN LCT LDDF LDLC LDLT LDMN LDMT LDPR LINE LOCT LRK MDS MDTJ MEDN MFR MID MOP MRP MSGF MSGL MSGN MST MTM MTO MTP MTY MVBY NME NOT OPL OPM OVL PDA PDAC PDAL PDAS PDS PID PLN PRF PSFX PSN PTP RBN RCF RDTJ RSNC RTBY RTM RTNC SCST SEC SID SLM SMI SMP SMU SOSJ SOSP SOST SSM SVCN TAC TVXP UNC VACT VCHG VDTJ VMV VRJ VRSL VTM V1 XDC XDTJ XTM X100 X200

Table 10. Structured Field Introducers by Subcommand (continued)

Subcommand	Structured Field Introducers
LISTDATASET	ABND BLKC BLKS BLKT CDTJ CJBN CLS CPGM CTLG CTM DC DD DEV DLRJ DLWJDPCT DSEQ DSN ENTN FILE LINE LPGM LRCL MC MSGL MSGN NME OWN RCFM RSNC RTDJ RTNC SC SG STEP SVCN SYS VJBN VNME VOL VRSR VSCD VSCN VTYP 2JBN 2NME 2SCD 2SCN
LISTOWNER	ADL DPT EMN EMU ENTN ETL FOR ITL LINE MSGL MSGN OWN RSNC RTNC SUR SVCN VLN
LISTPRODUCT	ENTN FCD LINE MSGL MSGN OWN PDSC PNME PNUM RCK RSNC RTNC SVCN VER VLN VOL
LISTRACK	ENTN LINE LOC MEDN MSGL MSGN RCK RSNC RST RTNC SVCN VOL
LISTVOLUME	ACN ACT ADTJ ATM AVL BIN BMN CDTJ CJBN CLS CTM CTNR DEN DESC DEST DLRJ DLWJ DSC DSN DSR ENTN FCD HLOC INTR LBL LCID LDEV LINE LOAN LOC LOCT LVC LVN MEDA MEDC MEDN MEDR MEDT MOVM MSGL MSGN MVS NLOC NME NVL OAC OBMN OBN OCE OLOC OWN OXDJ PEND PNUM PRD PVL PWT RBYS RCK RSNC RTDJ RTNC SDTJ SEQ SG STVC SVCN TRD TWT UID USEC USEM VAC VCAP VER VM VMIN VOL VOLT VPCT VRSI VRXI VST XDTJ
LISTVRS	DESC DDTJ ENTN LINE LOC MSGL MSGN NVRS OWN PRTY RET RSNC RTNC SC1 SVCN TYP VANX VDD VJBN VOL VRC VRS VRSI VRXI
SEARCHBIN	ENTN LINE LOC MEDN MSGL MSGN RCK RSNC RST RTNC SVCN VOL
SEARCHDATASET	CDTJ CTM DSN ENTN FILE KEYF KEYT LINE MSGL MSGN OWN RSNC RTNC SVCN VOL
SEARCHDATASET(EXPAND=YES)	The same SFIs as the LISTDATASET subcommand.
SEARCHPRODUCT	ENTN FCD LINE MSGL MSGN OWN PNME PNUM RSNC RTNC SVCN VER VLN VOL
SEARCHPRODUCT(EXPAND=YES)	The same SFIs as the LISTPRODUCT subcommand.
SEARCHRACK	ENTN LINE LOC MEDN MSGL MSGN RCK RSNC RST RTNC SVCN VOL
SEARCHVOLUME	ADTJ AVL DESC DEST DSC DSR ENTN HLOC INTR KEYF KEYT LBL LINE LOAN LOC LVC LVN MEDA MEDC MEDN MEDR MEDT MSGL MSGN OWN PEND RCK RSNC RTDJ RTNC SVCN TYPF TYPT VOL VST XDTJ
SEARCHVOLUME(EXPAND=YES)	The same SFIs as the LISTVOLUME subcommand.
SEARCHVRS	DDTJ ENTN LINE LOC MSGL MSGN NVRS OWN PRTY RET RSNC RTNC SVCN VANX VJBN VOL VRS VRSI VRXI
SEARCHVRS(EXPAND=YES)	The same SFIs as the LISTVRS subcommand.

Appendix C. DFSMSrmm Application Programming Interface Mapping Macros

DFSMSrmm API macros can be used to generate mappings: This section discusses:

- The parameter list generated by the list form of the EDGXCI macro as shown in "EDGXCI: Parameter List"
- The structured field definitions generated by the EDGXSF macro as shown in "EDGXSF: Structured Field Definitions" on page 76

EDGXCI: Parameter List

The mapping of the parameter list, which is generated by the list form of the EDGXCI macro, is a Product-sensitive Programming Interface.

The EDGXCI mapping macro is provided for information only. Although the fields and values of the parameter list are shown here, your application program should not directly access and modify the parameter list. Always use macro EDGXCI.

```
MYPL     DS    0D                ++ EDGXCI PARM LIST
MYPL_XVERSION DS XL1             ++ INPUT XVERSION
MYPL_XOPERATION DS XL1           ++  XOPERATION
MYPL_XOPERATION_BEGIN EQU 0      ++ XOPERATION.BEGIN KEYWORD
MYPL_XOPERATION_CONTINUE EQU 1   ++ XOPERATION.CONTINUE KEYWORD
MYPL_XOPERATION_RELEASE EQU 2    ++ XOPERATION.RELEASE KEYWORD
MYPL_XOPERATION_ENDALL EQU 3     ++ XOPERATION.ENDALL KEYWORD
MYPL_XOUTPUT DS XL1              ++  XOUTPUT
MYPL_XOUTPUT_LINES EQU 0         ++ XOUTPUT.LINES KEYWORD
MYPL_XOUTPUT_FIELDS EQU 1        ++ XOUTPUT.FIELDS KEYWORD
MYPL_XEXPAND DS XL1              ++  XEXPAND
MYPL_XEXPAND_YES EQU 0           ++ XEXPAND.YES KEYWORD
MYPL_XEXPAND_NO EQU 1            ++ XEXPAND.NO KEYWORD
MYPL_XAPIADDR DS A               ++   XAPIADDR
MYPL_XOUTBUFADDR DS A            ++   XOUTBUFADDR
MYPL_XSUBCMDADDR DS A            ++   XSUBCMDADDR
MYPL_XTOKEN DS CL4               ++   XTOKEN
MYPL_XRSV0001 DS CL8             ++ RESERVED  XRSV0001
MYPL_XRSV0002 DS CL4             ++ RESERVED  XRSV0002
MYPL_XRSV0003 DS CL8             ++ RESERVED  XRSV0003
MYPLL    EQU   *-MYPL            ++ LENGTH OF PLIST
```

Figure 56. Mapping of the Parameter List Using the List Form of EDGXCI

EDGXSF: Structured Field Definitions

Use macro EDGXSF in your application program to define the data that the DFSMSrmm API returns in your output buffer. This section includes:

- "EDGXSF Parameters"
- "EDGXSF Mapping"
- "EDGXSF Labeling Conventions" on page 77

EDGXSF Parameters

The EDGXSF parameters are:

DSECT=YES

DSECT=NO
An optional parameter that specifies whether a DSECT statement is generated. The default is DSECT=YES.

DSECT=YES
Indicates that a DSECT statement should be generated.

DSECT=NO
Indicates that a DSECT statement should not be generated.

,LIST=YES

,LIST=NO
An optional parameter that specifies whether the macro expansion is printed. The default is LIST=YES.

,LIST=YES
Indicates to print the expansion.

,LIST=NO
Indicates do not print the expansion.

,TITLE=YES

,TITLE=NO
An optional parameter that specifies whether the macro title is printed. The default is TITLE=YES.

,TITLE=YES
Indicates to print the title.

,TITLE=NO
Indicates do not print the title

EDGXSF Mapping

Always use macro EDGXSF to determine the exact labels used to define the DFSMSrmm SFIs. Figure 57 on page 77 shows the dummy control section and the data types that define the generic mapping for the SFIs defined in "Appendix A. Structured Field Introducers" on page 55.

```
         EDGXSF
*
* ********************************************************************
* *    Output Buffer
* ********************************************************************
*
XSF_OUTBUF           DSECT    Output Buffer
XSF_OUTBUF_BUFLNG    DS       1FL4    Buffer Length
XSF_OUTBUF_RQDLNG    DS       1FL4    Required Buffer Length
XSF_OUTBUF_DATALNG   DS       1FL4    Length of Output Data
XSF_OUTBUF_FIELDS    DS       0C      Start of Structured Fields
*
* ********************************************************************
* *    Structured Field Introducers for Structured Fields
* ********************************************************************
*
XSF_SFI         DSECT           Structured Field Introducers
XSF_SFI_LENGTH  DS   1FL2      Length
XSF_SFI_ID      DS   1CL0003   ID (identifier)
                ORG  XSF_SFI_ID
XSF_SFI_IDVAL   DS   1CL0002   ID (Identifier Value)
XSF_SFI_IDQUAL  DS   1CL0001   ID (Identifier Qualifier)
XSF_SFI_TYPE    DS   1FL1      Type
                DS   1CL0001   Reserved
XSF_SFI_DTYPE   DS   1FL1      Data type
XSF_SFI_LEN     EQU  *-XSF_SFI
XSF_SFI_DATA    DS   0C        Start of Data
*
* ********************************************************************
* *    Data Types (XSF_SFI_DTYPE)                                    **
* ********************************************************************
*
XSF_SFI_DTYPE_UNDEF     EQU  X'00' Undefined data
XSF_SFI_DTYPE_CHAR_FIX  EQU  X'01' n-byte character
XSF_SFI_DTYPE_BITFLAG   EQU  X'02' 1-byte bit flag byte (8 bits)
XSF_SFI_DTYPE_BIN8      EQU  X'03' 1-byte (hex) value
XSF_SFI_DTYPE_BIN15     EQU  X'04' 2-byte hex value
XSF_SFI_DTYPE_BIN31     EQU  X'05' 4-byte hex value
XSF_SFI_DTYPE_CHAR_VAR  EQU  X'07' Variable length character
XSF_SFI_DTYPE_JDATE     EQU  X'09' 4-byte packed decimal date yyyydddC
XSF_SFI_DTYPE_TIME      EQU  X'0A' 4-byte packed decimal time hhmmsstC
*
```

Figure 57. Mapping: Output Buffer and Structured Field Introducers

EDGXSF Labeling Conventions

This section includes the labeling conventions used in macro EDGXSF. The conventions are provided to assist you until such time as you are able to obtain macro EDGXSF.

Labeling: Begin and End Resource Groups

Resource groups, except for VOL and VRS, are defined using the following format:

- XSF_SFI_ID_xxxx and XSF_xxxx_LENGTH
- XSF_SFI_ID_Exxxx and XSF_Exxxx_LENGTH

Figure 58 on page 78 shows the ACCESS resource group.

```
* ********************************************************************
* **   Begin and End ACCESS                                         **
* ********************************************************************
XSF_SFI_ID_ACCESS  EQU X'021000'
XSF_ACCESS_LENGTH  EQU X'0008'
*
XSF_SFI_ID_EACCESS EQU X'021080'
XSF_EACCESS_LENGTH EQU X'0008'
```

Figure 58. Mapping of the Begin and End ACCESS Group

The VOL and VRS groups are defined using the following format:

- XSF_SFI_ID_xxx and XSF_xxxGRP_LENGTH
- XSF_SFI_ID_Exxx and XSF_ExxxGRP_LENGTH

Figure 59 shows an example of the VOL resource group.

```
* ********************************************************************
* **   Begin and End VOL                                            **
* ********************************************************************
XSF_SFI_ID_VOL     EQU X'036000'
XSF_VOLGRP_LENGTH  EQU X'0008'

XSF_SFI_ID_EVOL    EQU X'036080'
XSF_EVOLGRP_LENGTH EQU X'0008'
```

Figure 59. Mapping of the Begin and End VOL Group

Labeling: SFIs that Introduce Data

SFIs introduce data and are defined using the following format:

- XSF_SFI_xxxx_ID
- XSF_xxxx_LENGTH
- XSF_xxxx_DTYPE

Figure 60 shows an example of the ATM SFI.

```
* ********************************************************************
XSF_SFI_ATM_ID EQU X'806000' Assigned Time
XSF_ATM_LENGTH EQU X'000C'
XSF_ATM_DTYPE  EQU X'0A'
* ********************************************************************
```

Figure 60. Mapping of the ATM SFI

Labeling: Flags

Output data for some SFIs are defined as bit flags using the following format: XSF_xxxx_FLAG_name.

Figure 61 on page 79 shows an example of the ACT SFI.

```
* ******************************************************************
XSF_SFI_ACT_ID EQU X'802000' Actions on Release
XSF_ACT_LENGTH EQU X'0009'
XSF_ACT_DTYPE  EQU X'02'
*
XSF_ACT_FLAG_SCRATCH EQU X'80'
XSF_ACT_FLAG_REPLACE EQU X'40'
XSF_ACT_FLAG_INIT    EQU X'20'
XSF_ACT_FLAG_ERASE   EQU X'10'
XSF_ACT_FLAG_RETURN  EQU X'08'
XSF_ACT_FLAG_NOTIFY  EQU X'04'
* ******************************************************************
```

Figure 61. Mapping of the ACT SFI

Labeling: Bin(8) Data

Output data for some SFIs are defined as one-byte binary numbers using the following format: XSF_xxxx_DATA_name.

Figure 62 shows an example of the LOCT SFI.

```
* ******************************************************************
XSF_SFI_LOCT_ID EQU X'84E000' Location Type
XSF_LOCT_LENGTH EQU X'0009'
XSF_LOCT_DTYPE  EQU X'03'
*
XSF_LOCT_DATA_SHELF               EQU X'00'
XSF_LOCT_DATA_STORE_BUILTIN_BINS  EQU X'01'
XSF_LOCT_DATA_MANUAL              EQU X'02'
XSF_LOCT_DATA_AUTO                EQU X'03'
XSF_LOCT_DATA_STORE_BINS          EQU X'04'
XSF_LOCT_DATA_STORE_NOBINS        EQU X'05'
* ******************************************************************
```

Figure 62. Mapping of the LOCT SFI

Unlabeled Data

The following output data types are unlabeled:

- Fixed-length and variable-length character data
- Two-byte binary values
- Four-byte binary values
- Dates
- Times

Appendix D. Hexadecimal Example of an Output Buffer

This appendix provides an example and discussion of a hexadecimal representation of the contents of an output buffer for a SEARCHDATASET subcommand request. You can modify this example for use in your installation.

Hexadecimal Representation of an Output Buffer

Figure 63 is a hexadecimal representation of the contents in an output buffer that might be produced for the SEARCHDATASET VOLUME(VOL001) subcommand shown in “Requesting Standard Output” on page 31. The following format is used:

- Relative buffer address shown as 2-byte values.
- Buffer contents are shown in groups of 8-bytes.

```
0000 0000100000000000 0000007100080260 00000000001B82A0 00000007D6E6D5C5
0020 D9D6D5C54BC6C9C5 D3C44BE3C5E2E300 0E8BC000000001E5 D6D3F0F0F1001087
0040 0000000007D6E6D5 C5D9D6D5C5000C81 3000000009199711 7C000C81A0000000
0060 0A0815270C000C83 3000000005000000 0100080260800000 0000000000000000
0080 0000000000000000 0000000000000000 0000000000000000 0000000000000000

0FFC 0000000000000000 0000000000000000 0000000000000000 0000000000000000
0FFE 0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

Figure 63. Hexadecimal Representation of the Contents of an Output Buffer

Description of the Contents of an Output Buffer

The first line of the output buffer shown in Figure 63 shows:

```
0000 0000100000000000 0000007100080260 00000000001B82A0 00000007D6E6D5C5
```

- Three 4-byte length fields:

  00001000

  This is the length you specified for the output buffer.

  00000000

  This means that the output buffer is large enough. When the buffer length is too small, DFSMSrmm sets this field with the size of the buffer needed. DFSMSrmm also returns return code 108 and reason code 10.

  00000071

  This is the total size of the data in the output buffer, including the length of this field. You can use this data length to determine when there is no more data to process.

- Eight structured fields:

  0008026000000000

  This is the Begin DATASET group SFI, which begins at offset x’000C’ into the output buffer. Use this SFI to confirm that you are processing a DATASET SFI. When you do not want to process a group of structured fields, scan to the end of the group by looking for the corresponding End SFI, such as, the End DATASET group SFI in this example.

The first and second lines of the output buffer shown in Figure 63 on page 81 show:

```
0000 0000100000000000 0000007100080260 00000000001B82A0 00000007D6E6D5C5
0020 D9D6D5C54BC6C9C5 D3C44BE3C5E2E300 0E8BC000000001E5 D6D3F0F0F1001087
```

- Data Set Name structured field

  001B82A000000007 D6E6D5C5D9D6D5C54BC6C9C5D3C44BE3C5E2E3

  This is the Data Set Name structured field, which begins at offset x'0014' into the output buffer. The structured field consists of the 8-byte DSN SFI and, in this example, the 19-byte data set name (OWNERONE.FIELD.TEST). The length of the structured field is 27 bytes (8 plus 19) as shown by the x'001B' value at the beginning of the field.

- Volume Serial structured field

  000E8BC000000001 E5D6D3F0F0F1

  This is the Volume Serial structured field, which begins at offset x'002F' into the output buffer. The structured field consists of the 8-byte VOL SFI and the 6-byte volume serial (VOL001).

The second and third lines of the output buffer shown in Figure 63 on page 81 show:

```
0020 D9D6D5C54BC6C9C5 D3C44BE3C5E2E300 0E8BC000000001E5 D6D3F0F0F1001087
0040 0000000007D6E6D5 C5D9D6D5C5000C81 3000000009199711 7C000C81A0000000
```

- Owner structured field

  0010870000000007 D6E6D5C5D9D6D5C5

  This is the Owner structured field, which begins at offset x'003D' into the output buffer. The structured field consists of the 8-byte OWN SFI and the 8-byte owner (OWNERONE).

- Create Date structured field

  000C813000000009 1997117C

  This is the Create Date structured field, which begins at offset x'004D' into the output buffer. The structured field consists of the 8-byte CDTJ SFI and the 4-byte packed-decimal date (x'1997117C').

The third and fourth lines of the output buffer shown in Figure 63 on page 81 show:

```
0040 0000000007D6E6D5 C5D9D6D5C5000C81 3000000009199711 7C000C81A0000000
0060 0A0815270C000C83 3000000005000000 0100080260800000 0000000000000000
```

- Create Time structured field

  000C81A00000000A 0815270C

  This is the Create Time structured field, which begins at offset x'0059' into the output buffer. The structured field consists of the 8-byte CTM SFI and the 4-byte packed-decimal time (x'0815270C').

- Physical File Sequence structured field

  000C833000000005 00000001

  This is the Physical File Sequence structured field, which begins at offset x'0065' into the output buffer. The structured field consists of the 8-byte FILE SFI and the 4-byte binary sequence number (x'00000001').

- End DATASET group SFI

  0008026080000000

  This is the End DATASET group SFI, which begins at offset x'0071' into the output buffer.

Processing the Contents of an Output Buffer

To process the contents of an output buffer, consider using the following guidelines:

1. Base the XSF_OUTBUF definition in macro EDGXSF as shown in Figure 64 on the address of the output buffer you are interested in.

```
XSF_OUTBUF           DSECT    Output Buffer
XSF_OUTBUF_BUFLNG    DS       1FL4    Buffer Length
XSF_OUTBUF_RQDLNG    DS       1FL4    Required Buffer Length
XSF_OUTBUF_DATALNG   DS       1FL4    Length of Output Data
XSF_OUTBUF_FIELDS    DS       0C      Start of Structured Fields
```

Figure 64. Output Buffer Definition

2. Base the XSF_SFI definition in macro EDGXSF as shown in Figure 65 on the address of XSF_OUTBUF_FIELDS.

```
XSF_SFI         DSECT          Structured Field Introducers
XSF_SFI_LENGTH  DS   1FL2      Length
XSF_SFI_ID      DS   1CL0003   ID (identifier)
                ORG  XSF_SFI_ID
XSF_SFI_IDVAL   DS   1CL0002   ID (Identifier Value)
XSF_SFI_IDQUAL  DS   1CL0001   ID (Identifier Qualifier)
XSF_SFI_TYPE    DS   1FL1      Type
                DS   1CL0001   Reserved
XSF_SFI_DTYPE   DS   1FL1      Data type
XSF_SFI_LEN     EQU  *-XSF_SFI
XSF_SFI_DATA    DS   0C        Start of Data
```

Figure 65. SFI Definition

3. Find the type of structured field you are processing by using the two-byte structured field identifier at XSF_SFI_IDVAL.

   Note: The values of XSF_SFI_IDQUAL for ADL, address line SFI, and UID, User ID SFI, described in “Appendix A. Structured Field Introducers” on page 55 are not constant values.
4. Move to the next structured field by adding the length at XSF_SFI_LENGTH to the XSF_SFI pointer.
5. Verify that you have reached the end of the valid data in the output buffer by using the length of the output data at XSF_OUTBUF_DATALNG.
6. Determine the type of data you are processing, by using the value in XSF_SFI_DTYPE.
7. Obtain the length of the data that starts at XSF_SFI_DATA, by subtracting XSF_SFI_LEN from the structured field length at XSF_SFI_LENGTH. in the output buffer.
8. Move to the end of the SFI by adjusting the pointer. In this example, when your pointer is at offset x’00000071’ into the output buffer, there are two indicators that you are done with the contents of the buffer:
   - You are looking at the End DATASET group SFI.
   - Adjusting the XSF_SFI pointer by the length of this SFI (8 bytes) points you past the last byte of data in the buffer.
9. Repeat these steps to process each structured field.

In the examples shown in Figure 64 and Figure 65:

- Adding the length of the data (x'00000071') at XSF_OUTBUF_DATALNG to the address of XSF_OUTBUF_DATALNG results in the address just beyond the last byte of data in the output buffer. You might find this a useful double-check to ensure that you are looking at valid data.
- Your XSF_SFI pointer is at the first structured field in the output buffer (offset 000C in the buffer), and the SFI identifier value at XSF_SFI_IDVAL (0260) tells you that the SFI is a Begin DATASET group. To move to the next structured field, add XSF_SFI_LENGTH (0008) to your pointer.
- Your XSF_SFI pointer is now at the second structured field in the output buffer (offset 0014 in the buffer); XSF_SFI_IDVAL (82A0) identifies the SFI as DSN (Data Set Name); and XSF_SFI_LENGTH (001B) minus XSF_SFI_LEN (8) gives you a length of 19 bytes for the data set name. The type of data is variable-length character because the data type at XSF_SFI_DTYPE equals XSF_SFI_DTYPE_CHAR_VAR.

  One method to process SFIs is to use an SFI lookup table containing ID values and addresses of corresponding processing routines. Another method is to use the XSF_SFI_DTYPE: Call an appropriate data-type routine with the address of the SFI or SFI data and the address of an output area as inputs.

  After you finish processing this structured field, update the XSF_SFI pointer to the next structured field.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Information Enabling Requests
Dept. DZWA

5600 Cottle Road
San Jose, CA 95193 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Programming Interface Information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of DFSMSrmm.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States, or other countries, or both:

DFSMS/MVS
DFSMSrmm
IBM
MVS
RACF
SecureWay
OS/390

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines technical terms and abbreviations used in DFSMS documentation. If you do not find the term you are looking for, refer to the index of the appropriate DFSMS manual or view the *IBM Dictionary of Computing* located at:

http://www.ibm.com/networking/nsg/nsgmain.htm

This glossary includes terms and definitions from:

- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published part of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.
- The *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

The following cross-reference is used in this glossary:

See: This refers the reader to (a) a related term, (b) a term that is the expanded form of an abbreviation or acronym, or (c) a synonym or more preferred term.

A

abend. Abnormal end of task

AL. American National Standards Label

AMODE. Addressing mode

ANDVRS. An RMM ADDVRS TSO subcommand operand. See *Using AND*.

ANSI. American National Standards Institute

APAR. Authorized program analysis report

API. Application Programming interface

ASA. American Standards Association

assigned date. The date that the volume is assigned to the current owner. Assigned date is not meaningful for a scratch volume.

AUL. ANSI and user header or trailer label

automated tape library. A device consisting of robotic components, cartridge storage areas, tape subsystems, and controlling hardware and software, together with the set of tape volumes that reside in the library and can be mounted on the library tape drives. See also *tape library*. Contrast with *manual tape library*.

automatic cartridge loader. An optional feature of the 3480 Magnetic Tape Subsystem that allows preloading of multiple tape cartridges. This feature is standard in the 3490 Magnetic Tape Subsystem.

automatic recording. In DFSMSrmm, the process of recording information about a volume and the data sets on the volume in the DFSMSrmm control data set at open or close time.

availability. For a storage subsystem, the degree to which a data set or object can be accessed when requested by a user.

B

backup. The process of creating a copy of a data set or object to be used in case of accidental loss.

basic catalog structure (BCS). The name of the catalog structure in the integrated catalog facility environment. See also *integrated catalog facility catalog*.

BCS. See *basic catalog structure*.

bin number. The specific shelf location where a volume resides in a storage location; equivalent to a rack number in the removable media library. See also *shelf location*.

BLP. Bypass label processing

BTLS. Basic Tape Library Support

built-in storage location. One of the Removable Media Manager defined storage locations: LOCAL, DISTANT, and REMOTE.

C

cache fast write. A storage control capability in which the data is written directly to cache without using nonvolatile storage. Cache fast write is useful for temporary data or data that is readily recreated, such as the sort work files created by DFSORT. Contrast with *DASD fast write*.

cartridge eject. For an IBM 3494 Tape Library Dataserver, IBM 3495 Tape Library Dataserver, or an IBM Model M10 3495 Tape Library Dataserver the act of physically removing a tape cartridge usually under robot control, by placing it in an output station. The software logically removes the cartridge by deleting or updating the tape volume record in the tape configuration database. For a manual tape library dataserver, the act of logically removing a tape cartridge from the manual tape library dataserver by deleting or updating the tape volume record in the tape configuration database.

cartridge entry. For either an IBM 3494 Tape Library Dataserver, IBM 3495 Tape Library Dataserver, or a IBM Model M10 3495 Tape Library Dataserver, the process of logically adding a tape cartridge to the library by creating or updating the tape volume record in the tape configuration database. The cartridge entry process includes the assignment of the cartridge to scratch or private category in the library.

Cartridge System Tape. The base tape cartridge media used with 3480 or 3490 Magnetic Tape Subsystems. Contrast with *Enhanced Capacity Cartridge System Tape*.

cell. A single cartridge location within an automated tape library dataserver. See also *rack number*.

circular file. A type of file that appends data until full. Then, starting at the beginning of the file, subsequent incoming data overwrites the data already there.

command line. On a display screen, a display line usually at the bottom of the screen in which only commands can be entered.

concurrent copy. A function to increase the accessibility of data by enabling you to make a consistent backup or copy of data concurrent with the usual application program processing.

confirmation panel. A DFSMSrmm panel that lets you tell DFSMSrmm to continue or stop a delete or release action. You specify whether or not you want to confirm delete or release requests in your dialog user options.

container. A receptacle in which one or more exported logical volumes can be stored. A stacked volume containing one or more logical volumes and residing outside a virtual tape server library is considered to be the container for those volumes.

container volume. See *container*.

control data set. A VSAM key-sequenced data set that contains the complete inventory of your removable media library, as well as the movement and retention policies you define. In the control data set DFSMSrmm records all changes made to the inventory, such as adding or deleting volumes.

control data set ID. A one-to-eight character identifier for the DFSMSrmm control data set used to ensure that, in a multi-system, multi-complex environment, the correct management functions are performed.

convenience input. The process of adding a small number of tape cartridges to the IBM 3494 Tape Library Dataserver and IBM 3495 Tape Library Dataserver without interrupting operations, by inserting the cartridges directly into cells in a convenience input station.

convenience input/output station. A transfer station with combined tape cartridge input and output functions in the IBM 3494 Tape Library Dataservers only.

convenience input station. A transfer station, used by the operator to add tape cartridges to the IBM 3494 Tape Library Dataserver or an IBM 3495 Tape Library Dataserver, which is accessible from outside the enclosure area.

convenience output. The process of removing a small number of tape cartridges from the IBM 3494 Tape Library Dataserver or an IBM 3495 Tape Library Dataserver without interrupting operations, by removing the cartridges directly from cells in a convenience input station.

convenience output station. A transfer station, used by the operator to remove tape cartridges from the automated tape library dataserver, which is accessible from outside the enclosure area.

conversion. In DFSMSrmm, the process of moving your removable media library inventory from another media management system to DFSMSrmm. DFSMSrmm manages the inventory and policies once you have converted it.

create date. Create date for a dataset is the date that the dataset is written to tape. Create date can also be the date a data set was read if it was created before DFSMSrmm is in use. Create date is updated each time a data set is replaced and not extended. Create date for volumes and other resources defined to DFSMSrmm is the date the resource is defined to DFSMSrmm or the date specified on the command as the create date.

D

DASD. Direct access storage device

DASD fast write. An extended function of some models of the IBM 3990 Storage Control in which data is written concurrently to cache and nonvolatile storage and automatically scheduled for destaging to DASD. Both copies are retained in the storage control until the data is completely written to the DASD, providing data integrity equivalent to writing directly to the DASD. Use of DASD fast write for system-managed data sets is controlled by storage class attributes to improve performance. See also *dynamic cache management*. Contrast with *cache fast write*.

DASD volume. A DASD space identified by a common label and accessed by a set of related addresses. See also *volume*, *primary storage*, *migration level 1*, *migration level 2*.

data column. A vertical arrangement of identical data items, used on list panels to display an attribute, characteristic, or value of one or more objects.

data control block (DCB). A control block used by access method routines in storing and retrieving data.

data entry panel. A panel in which the user communicates with the system by filling in one or more fields.

Data Facility Storage Management Subsystem (DFSMS). An operating environment that helps automate and centralize the management of storage. To manage storage, SMS provides the storage administrator with control over data class, storage class, management class, storage group, and automatic class selection routine definitions.

Data Facility Sort. An IBM licensed program that is a high-speed data processing utility. DFSORT provides an efficient and flexible way to handle sorting, merging, and copying operations, as well as providing versatile data manipulation at the record, field, and bit level.

DCB. See *data control block*.

device. This term is used interchangeably with unit. You mount a tape on a unit or device, such as a 3490.

system-managed storage environment. An environment that helps automate and centralize the management of storage. This is achieved through a combination of hardware, software, and policies. In the system-managed storage environment for OS/390, the function is provided by DFSORT, RACF, and the combination of DFSMS and OS/390.

DFSMSdfp. A DFSMS functional component or base element of OS/390, that provides functions for storage management, data management, program management, device management, and distributed data access.

DFSMSdss. A DFSMS functional component or base element of OS/390, used to copy, move, dump, and restore data sets and volumes.

DFSMShsm. A DFSMS functional component or base element of OS/390, used for backing up and recovering data, and managing space on volumes in the storage hierarchy.

DFSMShsm-managed volume. (1) A primary storage volume, which is defined to DFSMShsm but which does not belong to a storage group. (2) A volume in a storage group, which is using DFSMShsm automatic dump, migration, or backup services. Contrast with *system-managed volume* and *DFSMSrmm-managed volume*.

DFSMShsm-owned volume. A storage volume on which DFSMShsm stores backup versions, dump copies, or migrated data sets.

DFSMSrmm. A DFSMS functional component or base element of OS/390, that manages removable media.

DFSMSrmm control data set. See *control data set*.

DFSMSrmm-managed volume. A tape volume that is defined to DFSMSrmm. Contrast with *system-managed volume* and *DFSMShsm-managed volume*.

disaster recovery. A procedure for copying and storing an installation's essential business data in a secure location, and for recovering that data in the event of a catastrophic problem. Compare with *vital records*.

DISTANT. A DFSMSrmm built-in storage location ID. See *built-in storage location*.

dual copy. A high availability function made possible by nonvolatile storage in some models of the IBM 3990 Storage Control. Dual copy maintains two functionally identical copies of designated DASD volumes in the logical 3990 subsystem, and automatically updates both copies every time a write operation is issued to the dual copy logical volume.

dump class. A set of characteristics that describes how volume dumps are managed by DFSMShsm.

duplexing. The process of writing two sets of identical records in order to create a second copy of data.

dynamic cache management. A function that automatically determines which data sets will be cached based on the 3990 subsystem load, the characteristics of the data set, and the performance requirements defined by the storage administrator.

E

EHPCT. Extended High Performance Cartridge Tape

eject. The process used to remove a volume from a system-managed library. For an automated tape library dataserver, the volume is removed from its cell location and moved to the output station. For a manual tape library dataserver, the volume is not moved, but the tape configuration database is updated to show the volume no longer resides in the manual tape library dataserver.

Enhanced Capacity Cartridge System Tape. Cartridge system tape with increased capacity that can only be used with 3490E Magnetic Tape Subsystems. Contrast with *Cartridge System Tape*.

entry panel. See *data entry panel*.

EREP. Environmental Record Editing and Printing program

expanded output. Expanded output occurs when you specify OUTPUT=FIELDS and EXPAND=YES. For those subcommands for which expanded output applies, your application program receives more variable data than for standard output.

expiration. The process by which data sets and volumes are identified as available for reuse. In DFSMSrmm, all volumes have an expiration date or retention period set for them either by vital record specification policy, by user-specified JCL when writing a data set to the volume, or by an installation default. When a volume reaches its expiration date or retention period, it becomes eligible for release.

expiration date. The date at which a file is no longer protected against automatic deletion by the system.

expiration processing. The process of inventory management that ensures expired volumes are released and carries out required release actions on those volumes.

export. The operation to remove one or more logical volumes from a virtual tape server library. First, the list of logical volumes to export must be written on an export list volume and then, the export operation itself must be initiated.

exported logical volume. A logical volume that has gone through the export process and now resides on a stacked volume outside a virtual tape server library.

export list volume. A virtual tape server logical volume containing the list of logical volumes to export.

external label. A label attached to the outside of a tape cartridge that is to be stored in an IBM 3494 Tape Library Dataserver or IBM 3495 Tape Library Dataserver. The label might contain the DFSMSrmm rack number of the tape volume.

extract data set. A data set that you use to generate reports.

F

field format. Field format is where the output consists of Structured Field Introducers and variable data rather than output in line format.

filtering. The process of selecting data sets based on specified criteria. These criteria consist of fully or partially-qualified data set names or of certain data set characteristics.

FIPS. Federal Information Processing Standard

FMID. Function modification identifier

FRR. Functional recovery routines

G

generation data group (GDG). A collection of data sets kept in chronological order. Each data set is a generation data set.

generation data set. One generation of a generation data group.

generation number. The number of a generation within a generation data group. A zero represents the most current generation of the group, a negative integer (-1) represents an older generation and, a positive integer (+1) represents a new generation that has not yet been cataloged.

GDG. See *generation data group*.

GDS. See *generation data set*.

giga (G). The information-industry meaning depends upon the context:

1. G = 1,073,741,824(2^{30}) for real and virtual storage
2. G = 1,000,000,000 for disk storage capacity (e.g., 4 Gb fixed disk)
3. G = 1,000,000,000 for transmission rates

GPR. General purpose register

GRS. Global resource serialization

guaranteed space. A storage class attribute indicating the space is to be preallocated when a data set is created. If you specify explicit volume serial numbers, SMS honors them. If space to satisfy the allocation is not available on the user-specified volumes, the allocation fails.

H

hardware configuration definition (HCD). An interactive interface in MVS that enables an installation to define hardware configurations from a single point of control.

HCD. See *hardware configuration definition.*.

high capacity input station. A transfer station, used by the operator to add tape cartridges to the IBM 3494 Tape Library Dataserver or IBM 3495 Tape Library Dataserver, which is inside the enclosure area.

high capacity output station. A transfer station, used by the operator to remove tape cartridges from the automated tape library dataserver, which is inside the enclosure area.

home. See *home location*.

home location. For DFSMSrmm, the place where DFSMSrmm normally returns a volume when the volume is no longer retained by vital records processing.

HPCT. High Performance Cartridge Tape

I

ICETOOL. DFSORT's multipurpose data processing and reporting utility.

ID. Identifier

IDRC. See *improved data recording capability*.

import. The operation to enter previously exported logical volumes residing on a stacked volume into a virtual tape server library. First, the list of logical volumes to import must be written on an import list volume and the stacked volumes must be entered, and then, the import operation itself must be initiated.

import list volume. A virtual tape server logical volume containing the list of logical volumes to import. This list can contain individual logical volumes to import and/or it can contain a list of stacked volumes in which all logical volumes on the stacked volume are imported.

imported logical volume. An exported logical volume that has gone through the import process and can be referenced as a tape volume within a virtual tape server library. An imported logical volume originates from a stacked volume that went through the export process.

improved data recording capability (IDRC). A recording mode that can increase the effective cartridge data capacity and the effective data rate when enabled and used. IDRC is always enabled on the 3490E Magnetic Tape Subsystem.

installation defined storage location. A storage location defined using the LOCDEF command in the EDGRMMxx parmlib member.

integrated catalog facility catalog. A catalog that is composed of a basic catalog structure (BCS) and its related volume tables of contents (VTOCs) and VSAM volume data sets (VVDSs). See also *basic catalog structure* and *VSAM volume data set*.

Interactive Storage Management Facility (ISMF). The interactive interface of DFSMS that allows users and storage administrators access to the storage management functions.

Interactive Problem Control System (IPCS). A system facility that allows interactive problem analysis.

Interactive System Productivity Facility (ISPF). An IBM licensed program used to develop, test, and run interactive, panel-driven dialogs.

internal label. The internal label for standard label tapes is recorded in the VOL1 header label, magnetically recorded on the tape media.

in transit. A volume is in transit when it must be moved from one location to another and DFSMSrmm believes that the move has started, but has not yet received confirmation that the move is complete. For a volume moving from a system-managed library, the move starts when the volume is ejected.

inventory management. The regular tasks that need to be performed to maintain the control data set. See also *expiration processing*, *storage location management processing*, and *vital record processing*.

IPCS. See *Interactive Problem Control System*.

IPL. Initial program load.

ISPF. See *Interactive System Productivity Facility*.

ISMF. See *Interactive Storage Management Facility*.

ISO. See *International Organization for Standardization*.

J

JCL. Job control language

JES2. Job entry subsystem 2

JES3. Job entry subsystem 3

JFCB. Job file control block

journal. A sequential data set that contains a chronological record of changes made to the DFSMSrmm control data set. You use the journal when you need to reconstruct the DFSMSrmm control data set.

K

keyword. A predefined word that is used as an identifier.

kilo (K). The information-industry meaning depends upon the context:

1. K = 1024(2^{10}) for real and virtual storage
2. K = 1000 for disk storage capacity (e.g., 4000 KB fixed disk)
3. K = 1000 for transmission rates

L

Library Control System. The Object Access Method component that controls optical and tape library operations and maintains configuration information.

Line Format. Line format is where text and variable data are formatted into lines suitable for displaying at a terminal or printing on hardcopy output.

LOCAL. A DFSMSrmm built-in storage location ID. See *built-in storage location*.

location name. A name given to a place for removable media that DFSMSrmm manages. A location name can be the name of a system-managed library, a storage location name, or the location *SHELF*, identifying shelf space outside a system-managed library or storage locations.

logical volume. Logical volumes have a many-to-one association with physical tape media and are used indirectly by MVS applications. They reside in a Virtual Tape Server or on exported stacked volumes. Applications can access the data on these volumes only when they reside in a Virtual Tape Server, which makes the data available via its tape volume cache or after the data has been copied to a physical volume through the use of special utilities.

low-on-scratch management. The process by which DFSMSrmm replenishes scratch volumes in a system-managed library when it detects that there are not enough available scratch volumes.

M

management class. A collection of management attributes that are defined by the storage administrator, used to control the release of allocated but unused space: to control the retention, migration, and backup of data sets: to control the retention and backup of aggregate groups, and to control the retention, backup, and class transition of objects. If assigned by ACS routine to system-managed tape volumes, it can be used to identify a DFSMSrmm vital record specification.

manual cartridge entry processing. The process by which a volume is added to the tape configuration database when it is added to a manual tape library dataserver. DFSMSrmm can initiate this process.

manual mode. An operational mode where DFSMSrmm runs without recording volume usage or validating volumes. The DFSMSrmm TSO commands, ISPF dialog, and inventory management functions are all available in manual mode.

manual tape library. A manual tape library is an installation-defined set of tape drives and the set of volumes that can be mounted on the drives. The IBM implementation includes one or more 3490 subsystems, each connected by a Library Attachment Facility to a processor running the Library Manager application, and a set of volumes, defined by the installation as part of the library, which resides in shelf storage located near the 3490 subsystems.

master system. The MVS system where the master DFSMSrmm control data set resides.

master volume. A private volume that contains data that is available for write processing based on the DFSMSrmm EDGRMMxx parmlib MASTEROVERWRITE operand.

media format. The type of volume, recording format and techniques used to create the data on the volume.

media library. See *removable media library*.

media management system. A program that helps you manage removable media. DFSMSrmm is a media management system.

media name. An up to 8 character value that describes the shape or type of removable media stored in a storage location. Examples of media name are: SQUARE, ROUND, CARTRDGE, 3480

media type. A value that specifies the volume's media type. Media type can be specified as: *, CST, ECCST, HPCT, or EHPCT.

MEDIA 1. cartridge system tape

MEDIA 2. enhanced capacity cartridge system tape

MEDIA 3. high performance cartridge tape

MEDIA 4. extended high performance cartridge tape

mega (M). The information-industry meaning depends upon the context:

1. M = 1,048,576(2^{20}) for real and virtual storage
2. M = 1,000,000 for disk storage capacity (e.g., 4000 MB fixed disk)
3. M = 1,000,000 for transmission rates

migration. The process of moving unused data to lower cost storage in order to make space for high-availability data. If you wish to use the data set, it must be recalled. See also *migration level 1* and migration level 2.

migration level 1. DFSMShsm-owned DASD volumes that contain data sets migrated from primary storage volumes. The data can be compressed. See also *storage hierarchy*. Contrast with *primary storage* and *migration level 2*.

migration level 2. DFSMShsm-owned tape or DASD volumes that contain data sets migrated from primary storage volumes or from migration level 1 volumes. The data can be compressed. See also *storage hierarchy*. Contrast with *primary storage* and *migration level 1*.

MVS image. A single occurrence of the MVS/ESA operating system that has the ability to process work.

N

name vital record specification. A vital record specification used to define additional retention and movement policy information for data sets or volumes.

NEXTVRS. An RMM ADDVRS TSO subcommand operand. See *Using Next*.

NL. No label

non-scratch volume. A volume that is not scratch, which means it has valid or unexpired data on it. Contrast with *scratch*.

NSL. Nonstandard label

O

OAM. See *object access method*.

object. A named byte stream having no specific format or record orientation.

object access method (OAM). An access method that provides storage, retrieval, and storage hierarchy management for objects and provides storage and retrieval management for tape volumes contained in system-managed libraries.

OPC/ESA. Operations Planning and Control/Enterprise Systems Architecture

optical volume. Storage space on an optical disk, identified by a volume label. See also *volume*.

optical disk. A disk that uses laser technology for data storage and retrieval.

option line. See *command line*

owner. In DFSMSrmm, a person or group of persons defined as a DFSMSrmm user owning volumes. An owner is defined to DFSMSrmm through an owner ID.

owner ID. In DFSMSrmm, an identifier for DFSMSrmm users who own volumes.

P

parallel. During conversion, when you install DFSMSrmm concurrently with an existing media management system, it is called running in parallel.

partitioned data set (PDS). A data set on direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

permanent data set. A user-named data set that is normally retained for longer than the duration of a job or interactive session. Contrast with *temporary data set*.

PF. Program function key

physical stacked volume. See *stacked volume*.

physical volume. Physical volumes have a one-to-one association with physical tape media and are used directly by MVS applications. They may reside in an automated tape library dataserver or be kept on shelf storage either at vault sites or within the data center where they can be mounted on stand-alone tape drives.

pool. A group of shelf locations in the removable media library whose rack numbers share a common prefix. The shelf locations are logically grouped so that the volumes stored there are easier to find and use.

pool ID. The identifier for a pool. You define pool IDs in parmlib member EDGRMMxx.

pooling. The process of arranging shelf locations in the removable media library into logical groups.

pool storage group. A type of storage group that contains system-managed DASD volumes. Pool storage groups allow groups of volumes to be managed as a single entity. See also *storage group*.

primary space allocation. Amount of space requested by a user for a data set when it is created. Contrast with *secondary space allocation*.

primary storage. A DASD volume available to users for data allocation. The volumes in primary storage are called primary volumes. See also *storage hierarchy*. Contrast with *migration level 1* and *migration level 2.*

primary vital record specification. The first retention and movement policy that DFSMSrmm matches to a data set and volume used for disaster recovery and vital record purposes. See also vital record specification and secondary vital record specification.

private tape volume. A volume assigned to specific individuals or functions.

protect mode. In protect mode, DFSMSrmm validates all volume requests.

pseudo-generation data group. A collection of data sets, using the same data set name pattern, to be managed like a generation data group. The ¬ masking character is used in DFSMSrmm to identify the characters in the pattern that change with each generation.

PSW. Program status word

PTF. Program temporary fix

pull list. A list of scratch volumes to be pulled from the library for use.

PUT. Program update tape

R

RACF. Resource Access Control Facility

rack number. A six-character identifier that corresponds to a specific volume's shelf location in the installation's removable media library, and is the identifier used on the external label of the volume to identify it. The rack number identifies the pool and the external volume serial number for a volume residing in an automated tape library dataserver. The rack number identifies the pool, the external volume serial, and shelf location number for a volume not residing in an automated tape library dataserver. The rack number is not written by the tape drive. It exists as an entry in the DFSMSrmm control data set and on the external label of the tape. See also *shelf location*

rack pool. A group of shelves that contains volumes that are generally read-only.

ready to scratch. This describes the condition where a volume is eligible for scratch processing while it resides in a storage location. Since no other release actions are required, the volume can be returned to scratch directly from the storage location.

recording format. For a tape volume, the format of the data on the tape; for example, 18 tracks or 36 tracks.

record-only mode. The operating mode where DFSMSrmm records information about volumes as you use them, but does not validate or reject volumes.

recovery. The process of rebuilding data after it has been damaged or destroyed, often by using a backup copy of the data or by reapplying transactions recorded in a journal.

relative start generation. Relative generation zero is the latest generation of a tape; Relative generation -1 is the previous generation of that tape. Relative generation -2 is the generation before the previous one.

REMOTE. A DFSMSrmm built-in storage location ID. See *built-in storage location*.

removable media. See *volume*.

removable media library. The volumes that are available for immediate use, and the shelves where they could reside.

Resource Access Control Facility (RACF). An IBM licensed program that provides for access control by identifying and verifying the users to the system; authorizing access to protected resources; logging the detected unauthorized attempts to enter the system; and logging the detected accesses to protected resources.

Resource Group. A collection of structured fields that describe the attributes of a resource such as a volume.

Restructured Extended Executor (REXX) Language. A general-purpose, high-level programming language, particularly suitable for EXEC procedures or programs for personal computing.

retention date. Retention date can be the date that a data set or volume is retained by a vital record specification or the date of the inventory management run when the data set or volume is no longer retained by a vital record specification.

retention period. The time for which DFSMSrmm retains a volume or data set before considering it for release. You can retain a data set or volume as part of disaster recovery or vital records management. You set a retention period through a vital record specification that overrides a data set's expiration date.

retention type. The types of retention for which DFSMSrmm retains a volume or data set before considering it for release. The retention types for data sets are BYDAYSCYCLE, CYCLES, DAYS, EXTRADAYS, LASTREFERENCEDAYS, UNTILEXPIRED, and WHILECATALOG. The retention types for volumes are DAYS and CYCLE.

REXX. Restructured Extended Executor Language

RMF. Resource Measurement Facility

RMM complex (RMMplex). One or more MVS images that share a common DFSMSrmm control data set.

RMODE. Residence mode

S

SAF. System Authorization Facility

scratch. The status of a tape volume that is available for general use, because the data on it is incorrect or is no longer needed. You request a scratch volume when you omit the volume serial number on a request for a tape volume mount.

scratch pool. The collection of tape volumes from which requests for scratch tapes can be satisfied. Contrast with *rack pool*.

scratch processing. The process for returning a volume to scratch status once it is no longer in use and has no outstanding release actions pending.

scratch tape. See *scratch volume*.

scratch volume. A tape volume that contains expired data only. See *scratch*.

SDB. Structured database

SDSF. Spool display and search facility

secondary space allocation. Amount of additional space requested by the user for a data set when primary space is full. Contrast with *primary space allocation*.

secondary vital record specification. The second retention and movement policy that DFSMSrmm matches to a data set and volume used for disaster recovery and vital records purposes. See also vital record specification and primary vital record specification.

SFI. See *structured field introducer*.

shelf. A place for storing removable media, such as tape and optical volumes, when they are not being written to or read.

shelf location. A single space on a shelf for storage of removable media. DFSMSrmm defines a shelf location in the removable media library by a rack number, and a shelf location in a storage location by a bin number. See also *rack number* and *bin number*

shelf-management. Is the function provided to manage the placement of volumes in individual slots in a location. Shelf-management is provided for the removable media library using rack numbers. For storage locations it is optional as defined by the LOCDEF options in parmlib and uses bin numbers.

shelf-resident volume. A volume that resides in a non-system-managed tape library.

shelf space. See *shelf*.

SL. Standard label

slot. See *shelf location*.

SMF. System management facility

SMP/E. System Modification Program Extended

stacked volume. Stacked volumes have a one-to-one association with physical tape media and are used in a Virtual Tape Server to store logical volumes. Stacked volumes are not used by MVS applications but by the Virtual Tape Server and its associated utilities. They may be removed from a Virtual Tape Server to allow transportation of logical volumes to a vault or to another Virtual Tape Server.

standard label. An IBM standard tape label.

Standard output. Standard output is the amount of variable data displayed, printed or put into a REXX variable in response to a subcommand. When you specify OUTPUT=LINES or EXPAND=NO with OUTPUT=FIELDS, your application program receives standard output as opposed to expanded output.

storage administrator. A person in the data processing center who is responsible for defining, implementing, and maintaining storage management policies.

storage class. A collection of storage attributes that identify performance goals and availability requirements, defined by the storage administrator, used to select a device that can meet those goals and requirements.

storage group. A collection of storage volumes and attributes, defined by the storage administrator. The collections can be a group of DASD volumes or tape volumes, or a group of DASD volumes and optical volumes treated as a single object storage hierarchy.

storage location. A location physically separate from the removable media library where volumes are stored for disaster recovery, backup, and vital records management.

(storage) location dominance. The priority used by DFSMSrmm to decide where to move a volume within the removable media library during vital record specification processing. It covers all the locations; SHELF, storage locations, and system-managed tape libraries.

storage location management processing. The process of inventory management that assigns a shelf location to volumes that have moved as a result of vital record processing. See also *vital record processing*

stripe. In DFSMS, the portion of a striped data set that resides on one volume. The records in that portion are not always logically consecutive. The system distributes records among the stripes such that the volumes can be read from or written to simultaneously to gain better performance. Whether it is striped is not apparent to the application program.

striping. A software implementation of a disk array that distributes a data set across multiple volumes to improve performance.

structured field. Output from the DFSMSrmm application programming interface consisting of a Structured Field Introducer and output data.

structured field introducer (SFI). An 8-byte entity that either introduces the beginning of a group of data or introduces output data that immediately follows the introducer.

subsystem. A special MVS task that provides services and functions to other MVS users. Requests for service are made to the subsystem through a standard MVS facility known as the subsystem interface (SSI). Standard MVS subsystems are the master subsystem and the job entry subsystems JES2 and JES3.

subsystem interface (SSI). The means by which system routines request services of the master subsystem, a job entry subsystem, or other subsystems defined to the subsystem interface.

SUL. IBM standard and user header or trailer label

SVC. Supervisor call

system-managed storage. Storage managed by the Storage Management Subsystem. SMS attempts to deliver required services for availability, performance, and space to applications. See also *system-managed storage environment*.

system-managed tape library. A collection of tape volumes and tape devices, defined in the tape configuration database. A system-managed tape library can be automated or manual. See also *tape library.*

system-managed volume. A DASD, optical, or tape volume that belongs to a storage group. Contrast with *DFSMShsm-managed volume* and *DFSMSrmm-managed volume*.

system programmer. A programmer who plans, generates, maintains, extends, and controls the use of an operating system and applications with the aim of improving overall productivity of an installation.

T

tape configuration database (TCDB). One or more volume catalogs used to maintain records of system-managed tape libraries and tape volumes.

tape librarian. The person who manages the tape library. This person is a specialized storage administrator.

tape library. A set of equipment and facilities that support an installation's tape environment. This can include tape storage racks, a set of tape drives, and a set of related tape volumes mounted on those drives. See also *system-managed tape library* and *automated tape library*.

Tape Library Control System (TLCS). IBM program offering 5785-EAW. DFSMSrmm replaces TLCS.

Tape Library Dataserver. A hardware device that maintains the tape inventory associated with a set of tape drives. An automated tape library dataserver also manages the mounting, removal, and storage of tapes. An automated or manual tape library that supports system-managed storage of tape volumes. IBM's automated tape library dataservers include the IBM 3494 Tape Library Dataserver and the IBM 3495 Tape Library Dataserver. IBM's manual tape library dataserver is the IBM Model M10 3495 Tape Library Dataserver.

tape storage group. A type of storage group that contains system-managed private tape volumes. The tape storage group definition specifies the system-managed tape libraries that can contain tape volumes. See also *storage group*.

tape subsystem. A magnetic tape subsystem consisting of a controller and devices, which allows for the storage of user data on tape cartridges. Examples of tape subsystems include the IBM 3490 and 3490E Magnetic Tape Subsystems.

tape volume. A tape volume is the recording space on a single tape cartridge or reel. See also *volume*.

TCDB. See *tape configuration database*.

temporary data set. An uncataloged data set whose name begins with & or &&, that is normally used only for the duration of a job or interactive session. Contrast with *permanent data set*.

tera (T). The information-industry meaning depends upon the context:

1. T = 1,099,511,627,776(2^{40}) for real and virtual storage
2. T = 1,000,000,000,000 for disk storage capacity (e.g., 4 TB of DASD storage)
3. T = 1,000,000,000,000 for transmission rates

TLCS. See *Tape Library Control System*.

TSO. Time Sharing Option

U

Until Expired. Allows the use of vital record specification policies for managing retention in a location as long as the volume expiration date has not been reached.

use attribute. (1) The attribute assigned to a DAD volume that controls when the volume can be used to allocate new data sets; use attributes are *public*, *private*, and *storage*. (2) For system-managed tape volumes, use attributes are *scratch* and *private*.

user volume. A volume assigned to a user, that can contain any data and can be rewritten as many times as the user wishes until the volume expires.

using AND. A method for linking DFSMSrmm vital record specifications to create chains of vital record specifications. DFSMSrmm applies policies in chains using AND only when all the retention criteria are true.

using NEXT. A method for linking DFSMSrmm vital record specifications to create chains of vital record specifications. DFSMSrmm applies policies in chains using NEXT one vital record at a time.

V

virtual export. Mark a volume as exported by using the DFSMSrmm subcommands.

virtual input/output (VIO) storage group. A type of storage group that allocates data sets to paging storage, which simulates a DASD volume. VIO storage groups do not contain any actual DASD volumes. See also *storage group*.

Virtual Tape Server (VTS). This subsystem, integrated into the Magstar 3494 Tape Library, combines the random access and high performance characteristics of DASD with outboard hierarchical storage management and virtual tape devices and tape volumes.

vital record group. A set of data sets with the same name that matches to the same DFSMSrmm vital record specification

vital record processing. The process of inventory management that determines which data sets and volumes DFSMSrmm should retain and whether a volume needs to move. These volumes and data sets have been assigned a vital record specification.

vital records. A data set or volume maintained for meeting an externally-imposed retention requirement, such as a legal requirement. Compare with *disaster recovery*.

vital record specification. Policies defined to manage the retention and movement of data sets and volumes used for disaster recovery and vital records purposes.

vital record specification management value. A one-to-eight character name defined by your installation and used to assign management and retention values to tape data sets. The vital record management value can be any value you chose to create a match between a vital record specification and data sets and volumes in your installation. By matching the vital record specifications to the data set or volumes, DFSMSrmm applies the retention and movement policies you define in the vital record specifications. During inventory management VRSEL processing, DFSMSrmm selects the correct, best matching vital record specification for a tape data set or volume.

VOLSER. See *volume serial number*.

volume. The storage space on DASD, tape, or optical devices, which is identified by a volume label. See also *DASD volume*, *logical volume*, *optical volume*, *stacked volume*, and *tape volume*.

volume catalog. See *tape configuration database*.

volume expiration date. The date the volume should expire based on the highest expiration date of the data sets that reside on the volume.

volume serial number (VOLSER). An identification number in a volume label that is assigned when a volume is prepared for use on the system. For standard label volumes, the volume serial number is the VOL1 label of the volume. For no label volumes, the volume serial number is the name the user assigns to the volume. In DFSMSrmm, volume serial numbers do not have to match rack numbers.

VTS. See *virtual tape server*.

W

warning mode. The operating mode in which DFSMSrmm validates volumes as you use them, but issues warning messages when it discovers errors instead of rejecting volumes.

write-to-operator (WTO). An optional user-coded service that allows a message to be written to the system console operator informing the operator of errors and unusual system conditions that may need to be corrected.

WTO. See *write-to-operator*.

Index

Numerics

A

B

C

D

E

F

N

O

P

R

S

T

U

V

X

Readers' Comments — We'd Like to Hear from You

OS/390
DFSMSrmm Application Programming Interface

Publication No. SC26-7332-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

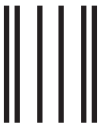
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC26-7332-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
RCF Processing Department
M86/050
5600 Cottle Road
SAN JOSE, CAU.S.A 95193-0001

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5647-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-7332-00