

z/OS



DFSMSrmm Application Programming Interface

z/OS



DFSMSrmm Application Programming Interface

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 121.

Tenth Edition, September, 2009

This edition applies to Version 1 Release 11 of z/OS® (5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC26-7403-08.

IBM® welcomes your comments. A form for readers’ comments may be provided at the back of this publication, or you may address your comments to the following address:

- International Business Machines Corporation
- Department 55JA, Mail Station P181
- 2455 South Road
- Poughkeepsie, NY 12601-5400
- United States of America
-
- FAX (United States & Canada): 1+845+432-9405
- FAX (Other Countries):
- Your International Access Code +1+845+432-9405
-
- IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)
- Internet e-mail: mhvrdfs@us.ibm.com
- World Wide Web: <http://www.ibm.com/servers/eserver/zseries/zos/webqs.html>

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this document
- Page number or topic related to your comment

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

© Copyright International Business Machines Corporation 1992, 2009.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures v

Tables vii

About this document ix

Required product knowledge ix
Referenced documents ix
Accessing z/OS DFSMS information on the Internet x
Using LookAt to look up message explanations. x
Notational conventions. xi
 How to read syntax diagrams xi
 How to abbreviate commands and operands xiii
 How to use continuation characters xiii
 Delimiters xiv
 Character sets xiv

Summary of Changes xv

Summary of Changes for SC26-7403-09 z/OS
Version 1 Release 11 xv
 New Information xv
 Changed Information xv
Summary of Changes for SC26-7403-08 z/OS
Version 1 Release 10 xvi
 New Information xvi
 Changed Information xvi
Summary of Changes for SC26-7403-07 z/OS
Version 1 Release 9 xvii
 New Information xvii
 Changed Information xvii
Summary of Changes for SC26-7403-06 z/OS
Version 1 Release 8 xvii
 New Information xvii
 Changed Information xviii

Chapter 1. Using the DFSMSrmm Application Programming Interface . . . 1

Supported RMM TSO Subcommands 2
Using the EDGXCI Macro 3
EDGXCI: Calling the DFSMSrmm Interface 3
 EDGXCI Environment 3
 EDGXCI Programming Requirements 3
 EDGXCI Syntax 5
 EDGXCI Parameters 6
 EDGXCI Return and Reason Codes 10
 EDGXCI Example 13

Chapter 2. Using the Object-Oriented DFSMSrmm Application Programming Interface Using C++ 17

DFSMSrmm High Level Language API Classes 23
 C++ classes 23
 Java class 23
DFSMSrmm API Methods 23
 Java Methods. 24

Receiving Extensible Markup Language (XML) Output Data in the XML Output Buffer 25

Chapter 3. Using the DFSMSrmm Application Programming Interface with Web Services. 29

Sample Java Web Service Client 31
Using Persistence and Parallel Processing 32
Defining How and When Authentication is Done. 32

Chapter 4. Using the DFSMSrmm Application Programming Interface Using Assembler Language. 33

Obtaining Resources 33
Specifying TSO Subcommand Input in the EDGXCI Macro 33
Using the CONTINUE Operation in the EDGXCI Macro 33
 Requesting multiple resources for SEARCH subcommands 34
Using Parameter Lists to Pass Information to the DFSMSrmm API. 35
 Coding a Single Parameter List, Single Token Area. 36
 Coding a Single Parameter List, Multiple Token Areas 38
 Coding Multiple Parameter List, Single Token Area. 40
 Coding Multiple Parameter List, Multiple Token Areas 40
Specifying the Option to Free a Resource 42
Specifying the Option to Release a Resource 43

Chapter 5. Using an Alternative Interface to the DFSMSrmm Application Programming Interface. 45

Parameter list to call EDGXHINT 46
Interface structure to pass the parameter list to EDGXHINT 47
Communication with the API 47
 Define the API 47
 Start API communication 48
 Issue a request 48
 Continue a request 48
 End a request. 49
 End API communication 49
Return and reason codes using EDGXHINT 49

Chapter 6. Processing the Output Data in the Output Buffer 51

Description of Structured Fields 51
Requesting SFI Data Format 52
 Requesting Line Format 52

Requesting Field Format	53
Requesting Types of Output	55
Requesting Standard Output.	55
Requesting Expanded Output	55
Accessing Return and Reason Codes	56
Accessing Messages and Message Variables.	57
Interpreting Date Format and Time Format.	57
Using Different Time Zones	58
Identifying Structured Field Introducers	58
Begin and End Resource Groups	58
System Return and Reason Code SFIs	59
Messages and Message Variables SFIs	60
SFIs for Output Data for Subcommands	61
ADD-Type of Subcommands.	62
CHANGE-Type of Subcommands	62
DELETE-Type of Subcommands	63
GETVOLUME Subcommand.	63
LIST-Type of Subcommands	63
LISTBIN SFIs	63
LISTCONTROL SFIs	64
LISTDATASET SFIs.	68
LISTOWNER SFIs	69
LISTPRODUCT SFIs	70
LISTRACK SFIs	70
LISTVOLUME SFIs.	71
LISTVRS SFIs.	73
SEARCH-Type of Subcommands	74
SEARCHBIN SFIs	74
SEARCHDATASET SFIs	75
SEARCHOWNER SFIs.	75
SEARCHPRODUCT SFIs	76
SEARCHRACK SFIs	76
SEARCHVOLUME SFIs	76
SEARCHVRS SFIs	77
Controlling Output from List and Search Type	
Requests	78
Limiting the Search for a Request	78
Output Buffer Examples	78

Appendix A. Structured Field Introducers 83

SFI Format	83
Structured Field Lengths	83
Compound SFI	84
SFIs for Begin and End Resource Groups	84
SFIs for Return and Reason Codes.	85
SFIs for Messages and Message Variables	86
SFIs for Subcommand Output Data	86

Appendix B. Structured Field Introducers by Subcommand 105

Appendix C. DFSMSrmm Application Programming Interface Mapping

Macros 109	
EDGXCI: Parameter List.	109
EDGXSF: Structured Field Definitions	110
EDGXSF Parameters	110
EDGXSF Mapping.	111
EDGXSF Labeling Conventions	112

Appendix D. Hexadecimal Example of an Output Buffer 115

Hexadecimal Representation of an Output Buffer	115
Description of the Contents of an Output Buffer	115
Processing the Contents of an Output Buffer	117

Appendix E. Accessibility 119

Using assistive technologies	119
Keyboard navigation of the user interface	119
z/OS information	119

Notices 121

Programming interface information	122
Policy for unsupported hardware.	122
Trademarks	122

Index 123

Figures

1. EDGXCI Macro Syntax Diagram	5	33. SFIs for CHANGEVOLUME with OUTPUT=FIELDS	62
2. Communicating with the DFSMSrmm API	14	34. SFIs for GETVOLUME with OUTPUT=FIELDS	63
3. Sample job control language (JCL) for Prelink Step	18	35. SFIs for LISTBIN with OUTPUT=FIELDS	64
4. Sample JCL for requesting LISTVOLUME information	19	36. SFIs for LISTCONTROL with OUTPUT=FIELDS	65
5. Sample Code for Using the High-Level Application Programming Interface	20	37. SFIs for LISTDATASET with OUTPUT=FIELDS	69
6. C++ Code Example for Writing XML Output to a File.	25	38. SFIs for LISTOWNER with OUTPUT=FIELDS	70
7. XMLFILE Output File	26	39. SFIs for LISTPRODUCT with OUTPUT=FIELDS	70
8. Example of Specifying the DFSMSrmm API Subcommand	33	40. SFIs for LISTRACK with OUTPUT=FIELDS	70
9. Example of Specifying the RMM TSO Subcommand	33	41. SFIs for LISTVOLUME with OUTPUT=FIELDS	72
10. Single Parameter List, Single Token Area	37	42. SFIs for LISTVRS with OUTPUT=FIELDS	74
11. Single Parameter List, Multiple Token Areas	38	43. SFIs for SEARCHBIN with OUTPUT=FIELDS,EXPAND=NO	75
12. Releasing All Resources	39	44. SFIs for SEARCHDATASET with OUTPUT=FIELDS,EXPAND=NO	75
13. Multiple Parameter Lists, Single Token Area	40	45. SFIs for SEARCHOWNER with OUTPUT=FIELDS,EXPAND=NO	75
14. Multiple Parameter Lists, Multiple Token Area	41	46. SFIs for SEARCHPRODUCT with OUTPUT=FIELDS	76
15. TOKEN= Specified on EDGXCI	43	47. SFIs for SEARCHRACK with OUTPUT=FIELDS,EXPAND=NO	76
16. TOKEN= Not Specified on EDGXCI	43	48. SFIs for SEARCHVOLUME with OUTPUT=FIELDS,EXPAND=NO	77
17. Binding a C++ program for use of EDGXHINT	45	49. SFIs for SEARCHVRS with OUTPUT=FIELDS,EXPAND=NO	77
18. C/C++ sample code for an interface struct	47	50. CONTINUE Example, First Output Buffer	79
19. Issue a TSO subcommand using EDGXHINT	48	51. CONTINUE Example, Second Output Buffer	80
20. Example of List Type of Output Using OUTPUT=LINES.	53	52. CONTINUE Example, Third (Last) Output Buffer	80
21. Example of Output Using OUTPUT=FIELDS	54	53. Mapping of the Parameter List Using the List Form of EDGXCI	109
22. Example of Search Type of Output Using EXPAND=NO.	55	54. Mapping of the Begin and End ACCESS Group	112
23. Example of Search Type of Output Using OUTPUT=FIELDS, EXPAND=YES	56	55. Mapping of the Begin and End VOL Group	113
24. Message and Message Variable Structured Fields	57	56. Mapping of the ATM SFI.	113
25. Begin and End Resource Group SFI Sequence	59	57. Mapping of the ACT SFI.	113
26. Begin and End Resource Group SFI Pairs	59	58. Mapping of the LOCT SFI	114
27. Begin and End Resource Group SFI Pairs for Subgroups	59	59. Hexadecimal Representation of the Contents of an Output Buffer	115
28. System Return and Reason Codes	60	60. Output Buffer Definition	117
29. SFIs for Messages and Message Variables	60	61. SFI Definition	117
30. Message Group with the CONT SFI	60		
31. Formatted Lines	62		
32. SFIs for ADDVOLUME with OUTPUT=FIELDS	62		

Tables

1. Character Sets	xiv	11. Parameter list for a call of EDGXHINT	46
2. Special Characters Used in Syntax.	xiv	12. Message Related SFIs	61
3. RMM TSO Subcommands	2	13. Begin and End Group Structured Field Introducers.	84
4. Return and Reason Codes for the EDGXCI Macro	11	14. Reason and Return Code SFIs	85
5. DFSMSrmm API Command C++ Classes	23	15. Message SFIs	86
6. DFSMSrmm API Command Java Class	23	16. Command SFIs	87
7. DFSMSrmm API C++ Methods	23	17. Structured Field Introducers by Subcommand	105
8. DFSMSrmm API Java Methods	24	18. Output Buffer and Structured Field Introducers	111
9. Return Codes and Reason Codes Issued when You Specify OPERATION=CONTINUE	34	19. Constants for EDGSFI.	112
10. Types of Parameter Lists	35		

About this document

This document is intended for application programmers who use the DFSMSrmm™ application programming interface to obtain information about resources that are managed by DFSMSrmm.

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, “Using the Object-Oriented DFSMSrmm Application Programming Interface Using C++,” on page 17 for information on using C++ and other high-level programming languages to write programs to obtain information about DFSMSrmm resources.
- Chapter 3, “Using the DFSMSrmm Application Programming Interface with Web Services,” on page 29 for information on using the DFSMSrmm application programming interface with Web services.
- Chapter 4, “Using the DFSMSrmm Application Programming Interface Using Assembler Language,” on page 33 for guidelines for using the application programming interface.
- Chapter 6, “Processing the Output Data in the Output Buffer,” on page 51 for information on the data that the DFSMSrmm application programming interface returns.

For information about accessibility features of z/OS, for users who have a physical disability, see Appendix E, “Accessibility,” on page 119.

Required product knowledge

To use this document effectively, you should be familiar with:

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

Referenced documents

These publications have additional information about DFSMSrmm:

Publication Title	Order Number
<i>z/OS DFSMSrmm Diagnosis Guide</i>	GY27-7619
<i>z/OS DFSMSrmm Managing and Using Removable Media</i>	SC26-7404
<i>z/OS DFSMSrmm Implementation and Customization Guide</i>	SC26-7405
<i>z/OS DFSMSrmm Reporting</i>	SC26-7406

This document also refers to the following publications:

Publication Title	Order Number
<i>z/OS XL C/C++ User's Guide</i>	SC09-4767
<i>z/OS Migration</i>	GA22-7499

Publication Title	Order Number
<i>z/OS Summary of Message and Interface Changes</i>	SA22-7505
<i>z/OS MVS System Messages, Vol 1 (ABA-AOM)</i>	SA22-7631
<i>z/OS MVS System Messages, Vol 2 (ARC-ASA)</i>	SA22-7632
<i>z/OS MVS System Messages, Vol 3 (ASB-BPX)</i>	SA22-7633
<i>z/OS MVS System Messages, Vol 4 (CBD-DMO)</i>	SA22-7634
<i>z/OS MVS System Messages, Vol 5 (EDG-GFS)</i>	SA22-7635
<i>z/OS MVS System Messages, Vol 6 (GOS-IEA)</i>	SA22-7636
<i>z/OS MVS System Messages, Vol 7 (IEB-IEE)</i>	SA22-7637
<i>z/OS MVS System Messages, Vol 8 (IEF-IGD)</i>	SA22-7638
<i>z/OS MVS System Messages, Vol 9 (IGF-IWM)</i>	SA22-7639
<i>z/OS MVS System Messages, Vol 10 (IXC-IZP)</i>	SA22-7640

Accessing z/OS DFSMS information on the Internet

In addition to making softcopy information available on CD-ROM, IBM provides access to z/OS softcopy information on the Internet. To view, search, and print z/OS information, go to the z/OS Internet Library:

<http://www.ibm.com/systems/z/os/zos/bkserv/>

Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM® messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can use LookAt from these locations to find IBM message explanations for z/OS® elements and features, z/VM®, VSE/ESA™, and Clusters for AIX® and Linux®:

- The Internet. You can access IBM message explanations directly from the LookAt Web site at www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/.
- Your z/OS TSO/E host system. You can install code on your z/OS systems to access IBM message explanations using LookAt from a TSO/E command line (for example: TSO/E prompt, ISPF, or z/OS UNIX® System Services).
- Your Microsoft® Windows® workstation. You can install LookAt directly from the *z/OS Collection* (SK3T-4269) or the *z/OS and Software Products DVD Collection* (SK3T-4271) and use it from the resulting Windows graphical user interface (GUI). The command prompt (also known as the DOS > command line) version can still be used from the directory in which you install the Windows version of LookAt.
- Your wireless handheld device. You can use the LookAt Mobile Edition from www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookatm.html with a handheld device that has wireless access and an Internet browser.

You can obtain code to install LookAt on your host system or Microsoft Windows workstation from:

- A CD in the *z/OS Collection* (SK3T-4269).
- The *z/OS and Software Products DVD Collection* (SK3T-4271).

- The LookAt Web site (click **Download** and then select the platform, release, collection, and location that suit your needs). More information is available in the LOOKAT.ME files available during the download process.

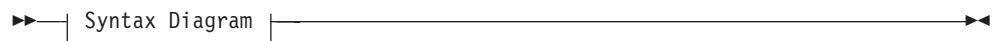
Notational conventions

This section explains the notational conventions used in this document.

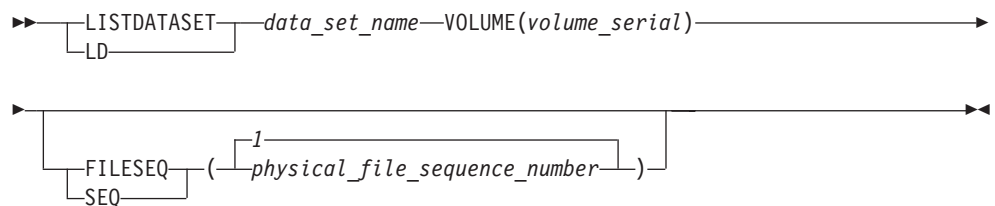
How to read syntax diagrams

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.



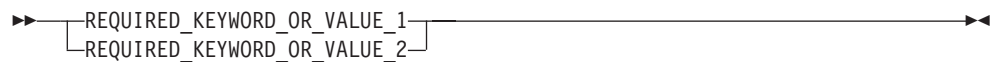
- If a diagram is longer than one line, each line to be continued ends with a single arrowhead and the next line begins with a single arrowhead.



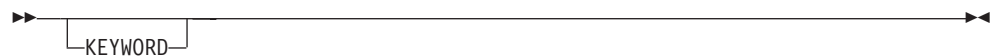
- Required keywords and values appear on the main path line. You must code required keywords and values.



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.



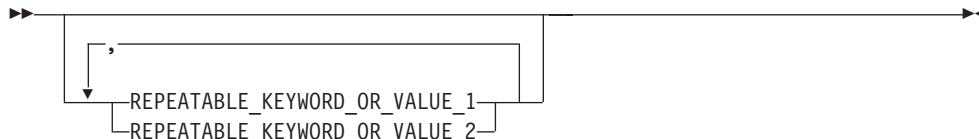
If several mutually exclusive optional keywords or values exist, they are stacked vertically in alphanumeric order below the main path line.



- 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.



Notes:

- 1 An example of a syntax note.
- References to syntax notes appear as numbers enclosed in parentheses above the line. Do not code the parentheses or the number.



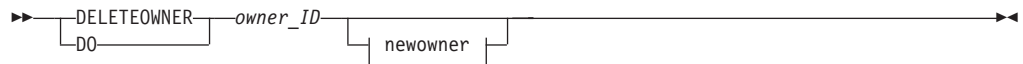
- Some diagrams contain *syntax fragments*, which serve to break up diagrams that are too long, too complex, or too repetitious. Syntax fragment names are in mixed case and are shown in the diagram and in the heading of the fragment. The fragment is placed below the main diagram.



Syntax Fragment:



The following is an example of a syntax diagram.



newowner



Notes:

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

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 document. 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'
<p>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'.</p>		

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
(b)	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
'	To enclose specified parameter values which contain special characters
(blank)	To delimit fields

Summary of Changes

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

You may notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

Summary of Changes for SC26-7403-09 z/OS Version 1 Release 11

This document contains information previously presented in *z/OS Version 1 Release 10 DFSMSrmm Application Programming Interface (SC26-7403-08)*.

The following sections summarize the changes to that information.

New Information

This edition contains the following new information:

- A new restriction has been added to “EDGXCI Restrictions” on page 4.
- “Requesting multiple resources for SEARCH subcommands” on page 34 has been added.
- Chapter 5, “Using an Alternative Interface to the DFSMSrmm Application Programming Interface,” on page 45 has been added.

This edition also includes new structured field introducers (see Table 16 on page 87):

- GDGC
- GDGD
- JRNT

Changed Information

The following sections have been updated to reflect the addition of the new MULTI=YES|NO parameter to the EDGXCI macro:

- “EDGXCI Syntax” on page 5
- Table 4 on page 11 has been updated to add return and reason codes for the MULTI=YES|NO parameter.
- “Using the CONTINUE Operation in the EDGXCI Macro” on page 33 has been updated to describe the effect of the MULTI=NO parameter.
- “SEARCH-Type of Subcommands” on page 74
- The EDGXCI parameter list in Figure 53 on page 109 has been updated to reflect changes for the MULTI=YES|NO parameter.

The descriptions of the following structured field introducers have changed (see Table 16 on page 87):

- X100
- X200
- X300

Table 17 on page 105 has been updated to reflect the new structured field introducers.

“EDGXSF Mapping” on page 111 has been updated with a new mapping.

Figure 45 on page 75 has been corrected.

Summary of Changes for SC26-7403-08 z/OS Version 1 Release 10

This document contains information previously presented in *z/OS Version 1 Release 9 DFSMSrmm Application Programming Interface (SC26-7403-07)*.

The following sections summarize the changes to that information.

New Information

Updated Chapter 3, “Using the DFSMSrmm Application Programming Interface with Web Services,” on page 29 with new information about the DFSMSrmm Web service.

This edition also includes new structured field introducers:

- DLTD for deleted by disposition processing.
- DSS6 for data set size.
- IRMM for Integrated Removable Media Manager.
- MEDINF for Begin and End resource group for MEDINF.
- MDNF for media information name.
- MDRA for MEDINF replace policy for age.
- MDRP for MEDINF replace policy for permanent errors.
- MDRT for MEDINF replace policy for temporary errors.
- MDRW for MEDINF replace policy for write mount count.
- MDRX for external recording technology.
- MDTX for external media type.
- USE6 for volume usage.
- VDRA for VRSDROP action.
- VDRC for VRSDROP count.
- VDRP for VRSDROP percent.
- VREA for VRSRETAIN action.
- VREC for VRSRETAIN count.
- VREP for VRSRETAIN percent.
- WORM for Write Once Read Many.
- XDRA for EXPDTRDOP action.
- XDRC for EXPDTRDOP count.
- XDRP for EXPDTRDOP percent.
- X300 for UX300 installation exit status.
- Many new SFIs for OPENRULE and PRITITION.

Changed Information

Changed web services info in Chapter 3, “Using the DFSMSrmm Application Programming Interface with Web Services,” on page 29.

Updated the following structured field introducers:

- AUD for SMF audit record type .
- MEDR for recording technology.
- MEDT for media type.
- SSM for SMF security record type .
- USEM for volume usage (KB).

- VACT for VRSMIN action.
- VCAP for volume/media capacity.

The structured field introducers for the LISTCONTROL suboperand in Table 17 on page 105 have been subdivided by parameter.

Summary of Changes for SC26-7403-07 z/OS Version 1 Release 9

This document contains information previously presented in *z/OS Version 1 Release 8 DFSMSrmm Application Programming Interface (SC26-7403-06)*.

The following sections summarize the changes to that information.

New Information

This edition includes new structured field introducers:

- CDSQ for control data set ENQ.
- CONT for SEARCH continue information.
- KEL1 for key encryption key label 1.
- KEL2 for key encryption key label 2.
- KEM1 for key encoding mechanism for key label 1.
- KEM2 for key encoding mechanism for key label 2.
- PRODVOL group for the LISTPRODUCT and SEARCHPRODUCT subcommands.
- PSF2 for second parmlib member suffix.

Changed Information

Updated the MEDR structured field introducer to include the recording format: EEFMT2.

Summary of Changes for SC26-7403-06 z/OS Version 1 Release 8

This document contains information previously presented in *z/OS Version 1 Release 7 DFSMSrmm Application Programming Interface (SC26-7403-05)*.

The following sections summarize the changes to that information.

New Information

Information added for the new TSO subcommand SEARCHOWNER.

New section added for “Using Different Time Zones” on page 58.

New methods added: `RmmTransaction.runCommandXmlShort()` and `RmmJApi.runCommandXmlShort()`. These methods return key data for the requested resources only, thus significantly reducing the size of the returned XML string.

This edition also includes new structured field introducers:

- For DFSMSrmm support for IBM TotalStorage Enterprise Tape System 3592.
- TZ and UTC for coordinated universal time (UTC), also known as GMT, support.
- DLR/DLRJ and TLR for vital record specification policy management simplification support.
- EML for DFSMSrmm enterprise level interface support.

Changed Information

Updated the following structured field introducers to include additional subcommands: RBYS, SC1, VRC, and VRSR.

Chapter 1. Using the DFSMSrmm Application Programming Interface

This topic tells you how to use the application programming interface (API) provided by DFSMSrmm (which is a z/OS feature) to read, extract, and update data in the DFSMSrmm control data set:

- From a high level language such as C++ or Java and receive the output through SFIs or XML
- Through Web services and receive the output through SFIs or XML
- From assembler language (using EDGXCI) and receive the output by line format or SFI format

You can use the output data to create reports or implement automation.

For details on using C++ or Web services, see these topics:

- Chapter 2, “Using the Object-Oriented DFSMSrmm Application Programming Interface Using C++,” on page 17
- Chapter 3, “Using the DFSMSrmm Application Programming Interface with Web Services,” on page 29

Use macro EDGXCI as described in “EDGXCI: Calling the 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 2 on page 14 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 110 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 “Requesting SFI Data Format” on page 52. You can also request standard output or expanded output as described in “Requesting Types of Output” on page 55.

To use the DFSMSrmm application programming interface, you must have High Level Assembler installed on your system. *z/OS 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 information
	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
SEARCHDATASET		SD	Create a list of data sets
SEARCHOWNER		SO	Create a list of owners
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 *z/OS DFSMSrmm Managing and Using Removable Media* for details on these subcommands.

Rule: 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: Calling the 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 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.

See Appendix C, "DFSMSrmm Application Programming Interface Mapping Macros," on page 109 for a complete description of the EDGXCI and EDGXSF macros.

EDGXCI Restrictions

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

The DFSMSrmm API uses Name/Token services to create a non-persistent task-level Name/Token pair for each TOKEN that has not been released. If you plan to use Checkpoint/Restart, refer to the section "Using Checkpoint/Restart with Name/Token Pairs" in *z/OS MVS Programming: Assembler Services Guide*, SA22-7605.

EDGXCI Input Register Information

Before issuing the EDGXCI macro, ensure that these 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:

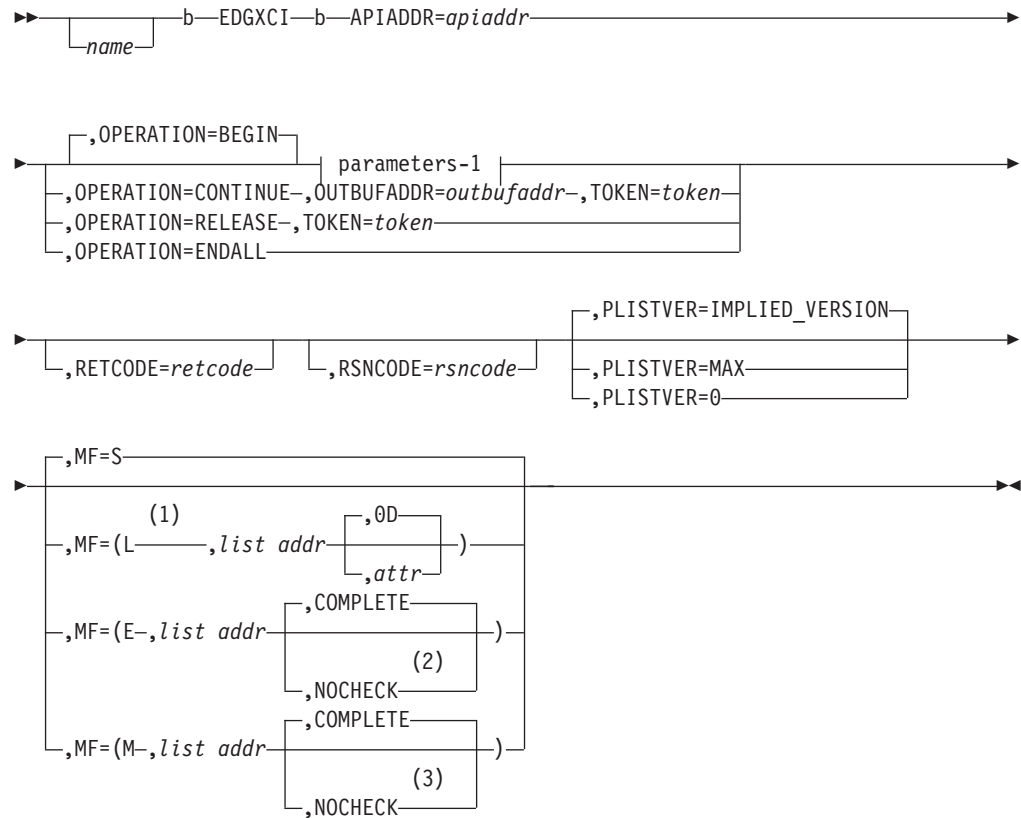
Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

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.

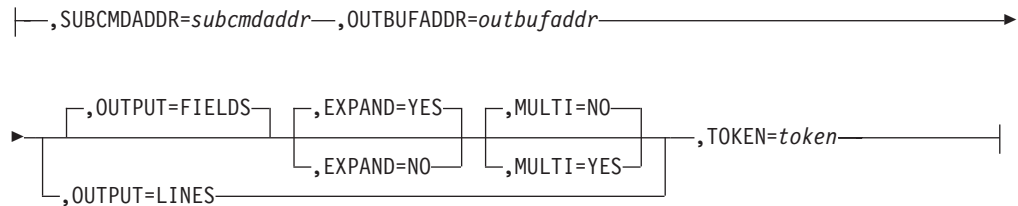
EDGXCI Syntax

Figure 1 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 1. 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 z/OS 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=NO

EXPAND=YES

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=NO

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

EXPAND=YES

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

Recommendation: Use the modify and execute forms of EDGXCI in this 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 ofX'0F'to force the parameter list to a word boundary orX'0D'to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value ofX'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.

MULTI=NO

MULTI=YES

When OUTPUT=FIELDS and OPERATION=BEGIN are specified, MULTI is an optional parameter that specifies whether a single resource group is to be returned in the buffer, or whether as many resources as fit in the buffer are to be returned. The default is MULTI=NO

MULTI=NO

Specifies that only a single entry can be handled by the API caller.

MULTI=YES

Specifies that multiple entries can be handled by the API caller.

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.

Recommendation: If you can tolerate the size change, 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 these:

- 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 these 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 *z/OS DFSMSrmm Managing and Using Removable Media* 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.
- When you use the API with high-level programming languages, DFSMSrmm returns a return code and reason code and a message described in the related messages column in Table 4. When you use the standard API, DFSMSrmm does not return a message but you can look to the related message for guidance.

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	Related Message
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.	EDG3900I
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.	EDG3901I
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	EDG3902E
108	04	Meaning: Required address of the input subcommand is missing. Action: You need to use SUBCMDADDR= subcmdaddr	EDG3903E

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

Return Code	Reason Code	Meaning and Action	Related Message
108	06	Meaning: Required address of your output buffer is missing. Action: Use OUTBUFADDR= outbufaddr to specify the parameter.	EDG3904E
108	08	Meaning: Your output buffer is less than 4096 bytes in size. Action: Obtain storage and set its length.	EDG3905E
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.	EDG3906E
108	12	Meaning: OPERATION parameter is invalid. Action: Use OPERATION= to specify the parameter; check your program for incorrect modifying of the parameter list.	EDG3907E
108	14	Meaning: OUTPUT parameter is invalid. Action: Use OUTPUT= to specify the parameter; check your program for incorrect modifying of the parameter list.	EDG3908E
108	16	Meaning: EXPAND parameter is invalid. Action: Use EXPAND= to specify the parameter; check your program for incorrect modifying of the parameter list.	EDG3909E
108	18	Meaning: MULTI parameter is invalid. Action: Use MULTI= to specify the parameter; check your program for incorrect modifying of the parameter list.	EDG3909E
108	56	Meaning: The token is already in use. Action: Use TOKEN= token to specify a token that is not in use.	EDG3910E
108	58	Meaning: OUTPUT=FIELDS is not supported for the subcommand specified by SUBCMDADDR= subcmdaddr . Action: Use OUTPUT=LINES or specify a different subcommand.	EDG3911E
108	60	Meaning: The length of the subcommand specified by SUBCMDADDR= subcmdaddr is too large. Action: Use a smaller subcommand.	EDG3912E
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.	

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

Return Code	Reason Code	Meaning and Action	Related Message
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.	EDG3913E
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.	EDG3914E
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.	EDG3915E
116	06	Meaning: An abnormal end has occurred. Action: Remove the cause of the abnormal end. Rerun your program.	EDG3916E
120	02	Meaning: Program error has occurred while you were using the high-level API. Action: Refer to the action provided with the specific reason code.	EDG3918E
120	04	Meaning: The LOAD for program EDGXAPI failed. Action: Correct the cause of the error and retry the command.	EDG3919E

EDGXCI Example

You can modify the example shown in Figure 2 on page 14to:

- 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 this 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.

Macro continuation characters must be entered in column 72.

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

```

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

```

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

Chapter 2. Using the Object-Oriented DFSMSrmm Application Programming Interface Using C++

DFSMSrmm Samples Provided in SAMPLIB

EDGHCLT is shipped in SAMPLIB. The sample code shows how to issue RMM subcommands by using the DFSMSrmm high-level language application programming interface classes and methods.

Requirement: The dynamic link library (DLL) is compiled using the IBM z/OS V1R10 XL C/C++ compiler. To compile your own program, you can use compiler versions up to and including the IBM z/OS V1R10 XL (ISO C/C++) level of the compiler.

Related reading: For information about using the IBM z/OS V1R10 XL C/C++ compiler, see *z/OS XL C/C++ User's Guide*. For migration and compatibility considerations, see *z/OS XL C/C++ Compiler and Run-Time Migration Guide for the Application Programmer*.

You can use C++ and other high-level programming languages to write programs to obtain information about DFSMSrmm resources. You use the same DFSMSrmm subcommand strings that you can use with the EDGXCI application programming interface. You can get output as structured field introducers or in Extensible Markup Language (XML). The XML output contains data and tags to define the data. DFSMSrmm provides a schema called `rmmxml.xsd` that contains the definitions for the XML. For XML output, DFSMSrmm converts the data to character in Unicode format as defined in the XML Schema file for the DFSMSrmm resources. See "Receiving Extensible Markup Language (XML) Output Data in the XML Output Buffer" on page 25.

To create your own program as shown in Figure 3 on page 18, you need access to the EDGXHCLU (header file) and the EDGXHCLL (definition side deck). The header file is necessary for the compile step and located in SYS1.MACLIB. The definition side deck is necessary for the bind step and is located in SYS1.SIEASID.

```

//COMPBIND JOB (4378),'COMPILE BIND HCLT',MSGCLASS=H,MSGLEVEL=(1,1),
//      TIME=3,CLASS=A,REGION=0M,NOTIFY=&SYSUID
//*
//*****
//*
//*   COMPILE AND BIND A C++ API USERPROGRAM
//*
//*****
//*
//*****
//* COMPILE STEP:
//* SYSLIB : LIBRARIES for C++ CLASS DEFINITION FILES AS SOURCE CODE
//*          INCLUDED IN THE USER PROGRAM, RMM HLL API CLASS
//*          DEFINITION FILE IS EDGXHCLU IN SYS1.MACLIB
//*****
I //COMPILE EXEC PGM=CCNDRVR,
//      PARM=(' /CXX OPTFILE(DD:CPARMS) '),
//      REGION=80M
//CPARMS DD *
//          XREF,OPTIMIZE,SOURCE,OBJ,MAR,
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR
//          DD DSN=CEE.SCEEH.H,DISP=SHR
//          DD DSN=CEE.SCEEH.SYS.H,DISP=SHR
I //SYSLIN DD DSN=&SYSUID..CPP.OBJ(EDGHCLT),DISP=SHR
I //SYSIN  DD DSN=&SYSUID..CPP.SOURCE(EDGHCLT),DISP=SHR
//*
//*****
//* BIND STEP:
I //* COMPILED MODULE EDGHCLT NEEDS TO BE CONCATENATED WITH DEFINITION *
//* SIDE DECK : SYS1.SIEASID(EDGXHCLL) SAME MEMBER NAME AS DLL
//*
I //* SYSLMOD : OUTPUT DATASET (HLQ.CPP.LOAD)HAS TO BE PDSE FORMAT *
//*****
//BINDCPP EXEC PGM=IEWL,REGION=1024K,
//      PARM='AMODE=31,MAP,RENT,DYNAM=DLL'
//SYSLIB DD DISP=SHR,DSN=CEE.SCEECP
//          DD DISP=SHR,DSN=CEE.SCEELKED
//SYSLMOD DD DISP=SHR,DSN=HLQ.CPP.LOAD
I //SYSLIN DD DISP=SHR,DSN=&SYSUID..CPP.OBJ(EDGHCLT)
//          DD DISP=SHR,DSN=SYS1.SIEASID(EDGXHCLL)
//          DD DDNAME=SYSIN
//SYSDEFSD DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSIN  DD *
//          NAME EDGHCLT(R) RC=0
//*

```

Figure 3. Sample job control language (JCL) for Prelink Step

Figure 4 on page 19 shows sample JCL that you can use to request information for the RMM LISTVOLUME subcommand.

```

/*-----*
/* JCL Example to use C/C++ HLLAPI submitting RMM LIST VOLUME command,*
/* using sample program EDGHCLT, *
/* EDGHCLT needs access to DLL: SYS1.SIEALNKE(EDGXHCLL) *
/* receiving SFI output (SFIFILE) and XML output (XMLFILE) *
/*-----*
//SMPLAPI EXEC PGM=EDGHCLT,PARM="'LISTVOLUME A00001'"
//STEPLIB DD DISP=SHR,DSN=HLQ.CPP.LOAD
//XMLFILE DD DISP=(NEW,CATLG),DSN=USERID.OUTPUT.XMLFILE,
//          UNIT=SYSALLDA,VOL=SER=RMMDSK,
//          SPACE=(CYL,(5,5)),DCB=(RECFM=VB,LRECL=1028,BLKSIZE=6144)
//SFIFILE DD DISP=(NEW,CATLG),DSN=USERID.OUTPUT.SFIFILE,
//          UNIT=SYSALLDA,VOL=SER=RMMDSK,
//          SPACE=(CYL,(5,5)),DCB=(RECFM=VB,LRECL=1028,BLKSIZE=6144)
//SYSPRINT DD SYSOUT=*

```

Figure 4. Sample JCL for requesting LISTVOLUME information

You need to write the program using C++ using the DFSMSrmm API classes and DFSMSrmm API methods to establish the connection to DFSMSrmm, issue the DFSMSrmm subcommands, and receive the output. If you select SFI format for the output, DFSMSrmm returns the information in structured field formats with all the fields provided.

Figure 5 on page 20 shows sample code that you can modify to use the high-level application programming interface.

```

| /*****
| *
| * Module Name:  EDGHCLT
| *
| * Description:  SAMPLE CODE for USING C/C++ HIGH LEVEL API INTERFACE *
| *
| *****/
| *
| * z/OS DFSMSrmm V1R11
| *
| * PROPRIETARY V3 STATEMENT
| * Licensed Materials - Property of IBM
| * 5694-A01
| * Copyright IBM Corp. 1993,2009
| * END PROPRIETARY V3 STATEMENT
| *****/
| *
| * Function:
| *
| *   This C++ Module is a sample program for the customer to use
| *   the High Level Language C/C++ API
| *
| *****/
| *
| * Change History
| *
| * $LV=RMMV1R6,1R6,030707 BRB: Created High Level API Interface   @LVA*
| *
| *****/
| #include <unistd.h>
| #include <stdlib.h>
| #include <stdio.h>
| #include <iostream.h>
| #include "EDGXHCLU"
|
| FILE* sfiFp;
|
| /*****
| * function to print SFI buffer into file
| *****/
| void printSFItoFile(RmmInterface::t_outp* outputPtr)
| {
|     int outputlen=outputPtr->header.out_used;
|     char* p = outputPtr->outputBuffer;
|     char ch;
|     int i,len = 0;
|     int offset = 0;
|     int l = 0;
|
|     for (l=0; l < outputlen; l++)
|     {
|         len = (*p * 16) + *(p+1);
|
|         if ( len == 0 ) break;
|
|         fwrite(p,1,len,sfiFp);
|
|         p = p + len;
|     }
| }

```

Figure 5. Sample Code for Using the High-Level Application Programming Interface (Part 1 of 3)


```

| /*****
| * start main
| *****/
| int main(int argc, char* argv [])
| {
|     long rc = 0;
|     FILE* xmlFp;
|     RmmApi* pApi;
|     RmmCommand* pCom;
|     char* tsoCommand;
|     tsoCommand = argv[1];
|
| /*****
| * get Output File names and open files
| *****/
|     if ( (xmlFp = fopen("DD:XMLFILE","w")) == NULL )
|     {
|         printf("could not open %s\n","DD:XMLFILE");
|         exit(0);
|     }
|     if ( (sfiFp = fopen("DD:SFIFILE","wb,type=record")) == NULL )
|     {
|         printf("could not open %s\n","DD:SFIFILE");
|         exit(0);
|     }
|
| /*****
| * create RmmApi object
| *****/
|     pApi = new RmmApi();
|     printf(" \nAPI object created \n");
|
| /*****
| * open Api
| *****/
|     if ( pApi->openApi() == 0 )
|     {
|         printf("API Return Code : %d\n",pApi->getApiRC());
|         printf("API Reason Code : %d\n",pApi->getApiRS());
|         printf("API Message      : %s\n",pApi->getMessageText());
|     }
|     else
|     {
|         printf("Could not open API \n");
|         exit(0);
|     }
|
| /*****
| * create RmmCommand object
| *****/
|
|     pCom = new RmmCommand(pApi);

```

Figure 5. Sample Code for Using the High-Level Application Programming Interface (Part 2 of 3)

```

| /*****
| * processes a TSO command
| *****/
|
| rc = pCom->issueCmd(tsoCommand);
|
| switch ( rc )
| {
|   case 0 :
|     printf("Return Code : %d\n",pCom->getApiRC());
|     printf("Reason Code : %d\n",pCom->getApiRS());
|     printf("Message      : %s\n",pCom->getMessageText());
|     printSFIToFile((RmmInterface::t_outp*) pCom->getBufferSfi());
|     fprintf(xmlFp,"%s\n",pCom->getBufferXml());
|     break;
|
|   case 1 :
|     printf("Return Code : %d\n",pCom->getApiRC());
|     printf("Reason Code : %d\n",pCom->getApiRS());
|     printf("Message      : %s\n",pCom->getMessageText());
|     printSFIToFile((RmmInterface::t_outp*) pCom->getBufferSfi());
|     fprintf(xmlFp,"%s\n",pCom->getBufferXml());
|
|     while( (pCom->getApiRC()==0) && (pCom->getApiRS()==4) )
|     {
|       rc = pCom->getNextEntry();
|       printf("Return Code : %d\n",pCom->getApiRC());
|       printf("Reason Code : %d\n",pCom->getApiRS());
|       printf("Message      : %s\n",pCom->getMessageText());
|       printSFIToFile((RmmInterface::t_outp*) pCom->getBufferSfi());
|       fprintf(xmlFp,"%s\n",pCom->getBufferXml());
|     }
|     break;
|
|   case -1:
|     printf("Return Code : %d\n",pCom->getApiRC());
|     printf("Reason Code : %d\n",pCom->getApiRS());
|     printf("Message      : %s\n",pCom->getMessageText());
|     break;
|
|   default:
|     printf("Return Code : %d\n",pCom->getApiRC());
|     printf("Reason Code : %d\n",pCom->getApiRS());
|     printf("Message      : %s\n",pCom->getMessageText());
| }
|
| /*****
| * destruction
| *****/
| delete pCom;
| delete pApi;
|
| fclose(sfiFp);
| fclose(xmlFp);
| exit(0);
| }
|
|                                     /* end main */

```

Figure 5. Sample Code for Using the High-Level Application Programming Interface (Part 3 of 3)

DFSMSrmm High Level Language API Classes

C++ classes

Use the DFSMSrmm RmmApi class to prepare the environment for using the RmmCommand class to use the DFSMSrmm TSO subcommands with the API. You can also use the RmmTransaction class that makes use of the RmmApi and RmmCommand classes. All of these classes are defined in the DFSMSrmm header file EDGXHCLU.

Table 5. DFSMSrmm API Command C++ Classes

Class	Description
RmmInterface	This is the superclass for DFSMSrmm processing. This class provides methods that are common to the classes RmmApi and RmmCommand. This class cannot be instantiated.
RmmApi	This class extends the RmmInterface class. Use this class to create an object to initiate a communication session with DFSMSrmm. You must create an instance of this class before you use class RmmCommand. This instance can be used to create one or more RmmCommand objects to enable you to run DFSMSrmm subcommands. You need one RmmApi object for each Multiple Virtual Storage (MVS) TCB under which DFSMSrmm runs. To end the communication session with DFSMSrmm and to no longer run subcommands, delete the RmmApi object.
RmmCommand	This class extends the RmmInterface class. Use this class to process a DFSMSrmm TSO subcommand. You must pass a reference to the RmmApi object when you instantiate an instance of this class. You can instantiate multiple instances of the RmmCommand class to process multiple commands in parallel. For example, you can use the output from a SEARCH command to issue LIST subcommands.
RmmTransaction	This class makes use of the RmmApi and RmmCommand classes. Instantiate an instance of this class, if you want to use the runCommandXml method.

Java class

If you want a Java™ application to access DFSMSrmm, use class RmmJApi.

Table 6. DFSMSrmm API Command Java Class

Class	Description
RmmJApi	Instantiate an instance of this class to communicate with DFSMSrmm from a Java application.

DFSMSrmm API Methods

Use the DFSMSrmm API methods to retrieve and update information about DFSMSrmm-managed resources. The naming convention for the methods is ClassName.methodName.

Table 7. DFSMSrmm API C++ Methods

Method	Description
RmmApi.openApi()	Use this method to check that DFSMSrmm is active and available to process commands.
RmmApi.closeApi()	Use this method when you no longer want to communicate with DFSMSrmm using this command session.
RmmCommand.issueCmd()	Use this method to issue a subcommand to DFSMSrmm. DFSMSrmm returns the subcommand return code and reason code. To access the output from the subcommand, use the getBufferSfi method or the getBufferXml method.
RmmCommand.getBufferSfi()	Use this method to obtain a string that contains the SFI output buffer from subcommand processing. Use this method after using the RmmCommand.issueCmd method and after using the RmmCommand.getNextEntry method.

Table 7. DFSMSrmm API C++ Methods (continued)

Method	Description
RmmCommand.getBufferXml()	Use this method to obtain a string that contains the XML output converted from the SFI output of subcommand processing.
RmmCommand.getNextEntry()	Use this method to retrieve information for the next resource or set of resources when there is more than one resource to be returned. For example, SEARCH subcommands and LISTCONTROL subcommands can return more than one resource. The getBufferXml and getBufferSfi methods can return multiple resources in a buffer; be sure to process all the returned data (XML or SFIs) before using the getNextEntry method if more entries may exist.
RmmInterface.getMessageText()	Use this method to obtain a string that contains the DFSMSrmm information or error message for the last command issued or the last getNextEntry method processing.
RmmInterface.getApiRc()	Use this method to obtain the return code from the last API request. Use the getMessageText method to retrieve the corresponding information or error message. See “EDGXCI Return and Reason Codes” on page 10 for information about message processing.
RmmInterface.getApiRs()	Use this method to obtain the reason code from the last API request. Use the getMessageText method to retrieve the corresponding information or error message. See “EDGXCI Return and Reason Codes” on page 10 for information about message processing.
RmmTransaction.runCommandXml()	Use this method to return a string containing the XML output converted from the SFI output of subcommand processing. It may also return error messages and return and reason codes for the command in the XML.
RmmTransaction.runCommandXmlShort()	Use this method to return a string containing the XML output for key values only. Only specific search commands return key fields. For example: <ul style="list-style-type: none"> • For SearchVolume, only the volser is returned. • For SearchDataset, only the datasetname, volume, and filesequence number are returned. • For SearchOwner, only the owner ID is returned. • For SearchRack/SearchBin, only the rack/bin number, location, and media name are returned. Other commands work as well, but they return all of the data, not just the key values.

Java Methods

Table 8. DFSMSrmm API Java Methods

Method	Description
RmmJApi.runCommandXml()	Use this method to return a string containing the XML output converted from the SFI output of subcommand processing. It may also return error messages and return and reason codes for the command in the XML.

Table 8. DFSMSrmm API Java Methods (continued)

Method	Description
RmmJApi.runCommandXmlShort()	<p>Use this method to return a string containing the XML output for key values only. Only specific search commands return key fields. For example:</p> <ul style="list-style-type: none"> • For SearchVolume, only the volser is returned. • For SearchDataset, only the datasetname, volume, and filesequence number are returned. • For SearchOwner, only the owner ID is returned. • For SearchRack/SearchBin, only the rack/bin number, location, and media name are returned. <p>Other commands work as well, but they return all of the data, not just the key values.</p>

Receiving Extensible Markup Language (XML) Output Data in the XML Output Buffer

Use the high-level language application programming interface to obtain output in XML format. The XML output may also return error messages and return and reason codes.

Figure 6 shows an example that issues an RMM SEARCHRACK subcommand and writes the XML output into the file named XMLFILE.

You can work with the output data in XML format by writing the output into a file or by parsing the output directly. You can define this file in the JCL, which you use to issue the command.

This example shows in C++ code how to:

- Issue a DFSMSrmm TSO subcommand by using the method `issueCommand()`.
- Use the method `getBufferXml()` to obtain access to the XML data.

```

FILE* xmlFp;                /* declare file pointer */
RmmApi* pApi;               /* declare an Api object */
RmmCommand* pCom;          /* declare a Command object */
pApi = new RmmApi();        /* create an Api object */
pApi->openApi();            /* open Api */
pCom = new RmmCommand(pApi); /* create a Command object */
pCom->issueCmd("SR RACK(*)"); /* issue a Command */
xmlFp = fopen("DD:XMLFILE","w") /* open the file for writing */
fprintf(xmlFp,"%s",pCom->getBufferXml()); /* print the data into the file */
fclose(xmlFp);              /* close the file */

```

Figure 6. C++ Code Example for Writing XML Output to a File

Figure 7 on page 26 shows the content of the file XMLFILE.

```

<?xml version="1.0" encoding="EBCDIC-CP-US" ?>
<document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="/usr/lib/xml_schema/rmmxml.xsd">
<RACK>
<RCK>RACK </RCK>
<VOL xsi:nil="true"></VOL>
<RST>EMPTY</RST>
<LOC>SHELF</LOC>
<MEDN>3480</MEDN>
<PID>*</PID>
</RACK>
<INFO>
<RTNC>4</RTNC>
<RSNC>4</RSNC>
<MSGT>EDG3011I 1 ENTRY LISTED </MSGT>
</INFO>
</document>

```

Figure 7. XMLFILE Output File

Most of the DFSMSrmm-produced XML tags use the SFI names described in Table 16 on page 87. For example, the XML tag for volume is <VOL>, which corresponds to the SFI name VOL. The DFSMSrmm-produced XML tags that do not use the SFI names are these tags.

- The XML tag <VOLINFO> for the volume resource group.
- The XML tag <VRSINFO > for the VRS resource group.
- The XML tags <JBN2>, <NME2>, <SCD2>, and <SCN2>, which represent the SFIs <2JBN>, <2NME>, <2SCD> and <2SCN>. XML does not allow tags to start with numeric characters.
- The XML tags <DSS6> and <USE6> are structured using additional tags for factor (<xxxxF>) and value (<xxxxS>), where xxxx is the XML tag name.

The XML output structure is declared in the XML schema file RMMXML.XSD, that you find in your file system directory /usr/lib/xml_schema. The schema contains type definitions for all elements.

The XML data stream contains a Uniform Resource Identifier (URI) to reference the required schema. To change the schema location, use the XML parser setExternalNoNamespaceSchemaLocation method.

DFSMSrmm ensures it creates only well-formed and valid XML documents and ensures that any text within an element contains only valid characters. The special characters &, <, >, ", and ' are escaped using the entities:

&amp;	&
&lt;	<
&gt;	>
&quot;	"
&apos;	'

Your XML parser will convert the entities back to the correct text character. Any code, such as CIM provider, that processes the XML document without a parser must consider that these entities might exist within the document and should be converted back to the correct character before use of the data.

Related Reading: You can write your own application to parse the XML data by using the XML parser. IBM provides an XML parser and sample applications in the XML Toolkit for z/OS available at

<http://www.ibm.com/zseries/software/xml> or from the IBM Software Delivery for System Modification Program Extended (SMP/E) installation.

Chapter 3. Using the DFSMSrmm Application Programming Interface with Web Services

DFSMSrmm Samples Provided

A sample Java Web service application, `rmmSampleWSClient.java`, is located in your file system directory `/usr/lpp/dfsms/rmm/`. The sample code shows how the application programming interface can be used with Web service.

Requirement: C/C++ or any other high-level language is required to exploit the DFSMSrmm class library. An XML parser (such as the one available in the XML toolkit for z/OS) is required to process the XML output from the DFSMSrmm application programming interface. Also, Language Environment for z/OS is required in order to install the DFSMSrmm class library. WebSphere Application Server for z/OS V6.0.2 and later, or an equivalent, is required to host the DFSMSrmm Web service. You can also use Apache Tomcat as an alternative web server, or another web service middleware. Rational Application Developer (RAD), formerly known as WebSphere Studio Application Developer, or an equivalent, is required for implementation and development. The minimum requirement to do any changes is Java SDK.

The Tomcat readme file, `rmmtc.txt`, located under `/usr/lpp/dfsms/rmm`, contains installation and setup information.

You can write Java applications that run on any platform that can use the DFSMSrmm API classes to obtain information about DFSMSrmm resources. You use the same DFSMSrmm subcommand strings that you can use with the EDGXCI application programming interface. You get output in Extensible Markup Language (XML). If you receive the output from the DFSMSrmm application programming interface as XML output, you can use an XML parser to process the returned data, or you can package the XML in order to use it as the base for displaying information for the end user. See “Receiving Extensible Markup Language (XML) Output Data in the XML Output Buffer” on page 25 for additional information about XML output data.

Using Web services, the DFSMSrmm application programming interface appears to the application as a local application programming interface even though it is running on another system. The infrastructure to support the use of Web services must be implemented and available on both the application system and the target z/OS system running DFSMSrmm. The infrastructure to support Web services on the target z/OS system is provided by WebSphere Application Server or Apache Tomcat open source servlet container. You can use an equivalent product, but additional customization and programming may be required by you. You can use Rational Application Developer (RAD) to develop applications that use the DFSMSrmm application programming interface with Web services.

The DFSMSrmm application programming interface Web service can be deployed either under z/OS WebSphere Application Server or Apache Tomcat (or another web service middleware).

The web service to be used under z/OS WebSphere Application Server is an Enterprise ARchive (EAR) file called rmmapi.ear and is located in your file system directory /usr/lpp/dfsms/rmm/. This EAR file contains all the elements needed to implement and use the Web service. To install the DFSMSrmm Web service, use the WebSphere Install Application. You can use either the graphical user interface or the command line tool for the installment and customization of your WebSphere environment. To develop a client application that uses the DFSMSrmm Web service, either import the EAR file into your project using Rational Application Developer (RAD) and use the definitions and codes it contains for your application, or use the sample client application shipped with DFSMSrmm, rmmSampleWSClient.java. After your application is written, modify the installation or environment-dependent information in the EAR file so you can implement the Web service in your environment. For more information, see the general web service help file rmmwebs.txt.

The web service to be used under Apache Tomcat is shipped as a Web ARchive (WAR), called rmmapi.war. For more information, see the general web service help file rmmwebs.txt. An additional help file for the Apache Tomcat environment, rmmtc.txt, is also available.

The Java class, RmmJApi.class, is the core part of the DFSMSrmm Web services. You can use it to access DFSMSrmm from inside z/OS, too. Packaged in rmmjapi.jar, located in your file system directory /usr/lpp/dfsms/rmm/, it is available to access the DFSMSrmm application programming interface locally from a Java program. It is important to make sure that the rmmjapi.jar file is included in the CLASSPATH environmental variable. RmmJApi.class supports the method RmmJApi.runCommandXml. See “DFSMSrmm High Level Language API Classes” on page 23 for additional information.

When you use the runCommandXml method to run a search command, it is possible to encounter a memory size limitation problem. A default limit of one megabyte is set for the returned data. This equals roughly 500 volumes (one volume resulting in about 2 kilobytes of data). If you are requesting a larger number of resources to be returned, you will reach this limit. (See the readme files for information on how to increase the memory limit of 1 megabyte.) The returned XML string ends after a complete resource, and message EDG3921I is added to the string. This message explains system status. Additionally, return code 4 and reason code 10 are added to enable you to correctly handle the returned data. You can narrow the search request by using one or more of the operands on the search subcommand, such as LIMIT, OWNER, or CONTINUE, or try to adjust the default limit (see *z/OS DFSMSrmm Implementation and Customization Guide* for additional information). The possible maximum limit depends on your environment. Check your JVM (Java Virtual Machine) and TCPIP settings. Using the CONTINUE operand, you can issue a sequence of calls to the web service, with the second and subsequent requests including the continue information returned by the previous request.

Another way to deal with memory size limitation is to use method runCommandXmlShort (see “DFSMSrmm API Methods” on page 23). This method returns key data for the requested resources only, thus significantly reducing the size of the returned XML string.

To further help with memory usage and to reduce the amount of data returned from the Web service, you can use GZIPInputStream to zip the command string and then you can use GZIPOutputStream to convert the returned output back to a string. See rmmSampleWSClient.java for a coding example.

You may want to publish your DFSMSrmm application programming interface Web service in a UDDI registry. The sample client comes without UDDI support. It is your task to publish the Web service to an UDDI registry and to implement the code for the discovery of the service. You can also write your application so that it does not need to dynamically discover where the Web service is located, or you can use a local or more general UDDI registry to discover the system that provides the Web service you need. If the services that the DFSMSrmm application programming interface Web service provides are specific only to your local system, it is recommended that you use a UDDI registry that is local to your system.

Sample Java Web Service Client

The sample client code needs to be compiled with a Java compiler (javac) to obtain the executable application. It contains:

- Some general methods to handle the Web Service endpoint and create a call.
- A client-side method to access the Web Service method runCommandXmlZip() communicating with byte arrays.
- A client-side method to access the Web Service method runCommandXML() communicating with strings arrays.
- A main program that:
 - Handles the passed command line parameters.
 - Zips the TSO subcommand to a byte array.
 - Creates a client object.
 - Sets the end point.
 - Calls the Web service.
 - Unzips the results and optionally writes to a file.
 - For reference, there is code that shows how to pass both commands and data as strings.

Usage:

```
java rmmSampleWSClient -i ip_address [-p port] [-u userid:password] [-d]
[-o output_file] [-x xml_schema] [-svwz] command
```

where:

```
-i = IP-address or domain name of the remote server
-p = Port number of the web service (default: 8080)
-u = Authorized user credentials, separated by a colon (default: none)
-o = Output file name (default: Screen output)
-d = Debug mode, for network connection test only
-x = XML schema file to be used for validation (default: No validation)
-s = Short XML response (default: Long XML response)
-v = Verbose mode On (default: Off)
-w = Use WebSphere server (default: Use Tomcat server)
-z = Zipped request (default: Unzipped request)
command = A valid DFSMSrmm TSO subcommand,
for example, LISTCONTROL OPTION
```

A sample Java web service client, EDGSJWS1, is provided in /usr/lpp/dfsms/rmm/rmmSampleWSClient.java. For information on how to use the DFSMSrmm Web service sample client, see the *z/OS DFSMSrmm Implementation and Customization Guide*.

Using Persistence and Parallel Processing

The Web service uses a stateless session bean and enables a single command to be run and the output returned in a single request. The method `RmmJApi.runCommandXml` enables a command to be run by a single method call. See Table 8 on page 24 for additional information.

Each caller of the Web service can use a different bean in WebSphere, and this enables multiple commands to be run in sequence and also in parallel. By customizing implementation options, you can enable WebSphere to instantiate a stateless session bean to support the DFSMSrmm Web service and to retain the session bean for use by any Web service requests. You can also limit how many instances of the bean can be running at one time.

Defining How and When Authentication is Done

Authentication is not done by the DFSMSrmm Web service. You must use the capabilities provided by the web service server to define how and when authentication is done. All DFSMSrmm subcommands use the RACF ACEE to perform authorization checking before the subcommand is processed. Therefore, ensure that the authentication performed by web services causes a valid ACEE to be created and that ACEE represents a valid RACF userid in the z/OS environment.

When using WebSphere, you must use the capabilities provided by WebSphere Application Server to define how and when authentication is done. All DFSMSrmm subcommands issued using the DFSMSrmm application programming interface from within WebSphere uses the RACF ACEE to perform authorization checking before the subcommand is processed. Therefore, ensure that the authentication performed using WebSphere causes a valid ACEE to be created and that ACEE represents a valid RACF userid in the z/OS environment. At a minimum, ensure that WebSphere is configured to:

- Perform basic authentication.
- Ensure that the extension and binding files for both client and server requests and responding security settings match.
- Provide your chosen authentication method.

When using Apache Tomcat, the Tomcat server must be configured for RACF/SAF Authentication and Authorization by downloading a separate package called "Tomcat SAF Security 5.5" from www.dovetail.com/downloads/jzos/index.html. The applied security model is called the Declarative Security, which is the expression of application security external to the application. It allows runtime configuration of application security without re-coding the application.

The web application configures Declarative Security in its unique deployment descriptor, `web.xml`. This is a required XML-formatted configuration file (also called the deployment descriptor) found in each web application's `WEB-INF` directory. Tomcat uses role-based authorization to manage access. With this model, access permissions are granted to an abstract entity called a security role, and access is allowed only to users or groups of users, who have that role. The deployment descriptor specifies the type of access granted to each role, but does not specify the role to user or group mappings. That's done in the user repository, which is typically another XML-formatted file in the server's production environment. See the Tomcat readme file, `rmmtc.txt`, for information on how to customize the XML-files for RACF/SAF-based security.

Chapter 4. Using the DFSMSrmm Application Programming Interface Using Assembler Language

Use the general programming guidelines to help you write your application program.

Obtaining Resources

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

To use resources most efficiently, consider these items.

- Use a different output buffer for each RMM TSO subcommand request. Reuse an output buffer to begin a new subcommand request only when there is nothing in the buffer that you need.
- Allocate a sufficient number of token areas, and parameters lists.
- Use the correct token when continuing a RMM TSO subcommand or when releasing a particular set of resources.
- Reuse a token to begin a new RMM TSO subcommand only when you no longer need the information obtained from the previous request.
- Reuse the resources associated with the token, especially when you are processing hundreds or thousands of subcommands.

Specifying TSO Subcommand Input in the EDGXCI Macro

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 8.

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

Figure 8. Example of Specifying the DFSMSrmm API Subcommand

Do not specify it as an RMM command with multiple input lines, as shown in Figure 9.

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

Figure 9. 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.

Using the CONTINUE Operation in the EDGXCI Macro

Use the EDGXCI OPERATION=CONTINUE parameter in your application program to ensure that you obtain all the available data. When you use OPERATION=CONTINUE, you might not receive more output data or you might receive only messages in your output buffer.

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 that uses a SEARCH command with OUTPUT=FIELDS and MULTI=NO is specified (or assumed by default).
- There is no more data to return to your application program.

The DFSMSrmm API issues return codes and reason codes indicating the results of processing when you specify OPERATION=CONTINUE. Write your application program to check the return codes and reason codes that the DFSMSrmm API returns to your application program.

Table 9. Return Codes and Reason Codes Issued when You Specify OPERATION=CONTINUE

Return Code	Reason Code	Processing
0	0	DFSMSrmm issues this return code and reason code in response to a search type subcommand. DFSMSrmm will not return any more records because there are no more records to return or because the search limit has been reached.
0	4	DFSMSrmm issues this return code and reason code when you issue requests specifying the LISTCONTROL subcommand and there are more records to return. Specify the OPERATION=CONTINUE to obtain more records.
4	2	DFSMSrmm issues this return code and reason code 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 but there might be more records to return.
4	4	DFSMSrmm issues this return code and reason code in response to a search type subcommand. The DFSMSrmm API issues these codes when the search processing indicates fewer records returned than were requested.
4	8	DFSMSrmm issues this return code and reason code in response to a search type subcommand. The DFSMSrmm API issues these codes when no entry meets then search criteria during search processing.

See “Controlling Output from List and Search Type Requests” on page 78 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.

Requesting multiple resources for SEARCH subcommands

The DFSMSrmm API can return resources either one at a time or multiple at a time when you specify one of the DFSMSrmm TSO RMM SEARCHDATASET, SEARCHBIN, SEARCHOWNER, SEARCHPRODUCT, SEARCHRACK, SEARCHVOLUME, and SEARCHVRS subcommands together with OUTPUT=FIELDS. Use the MULTI keyword to notify the API about which type of output you can handle. To specify MULTI=YES, your application must be able to

handle multiple resources each separated by the begin/end group SFIs. When you specify MULTI=YES, your output buffer can have one or more resource groups returned in a single call of the API. Using MULTI=YES helps reduce the system resources used for API processing.

Using Parameter Lists to Pass Information to the DFSMSrmm API

You can write your application program to include this processing:

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

You can use variations of parameter lists and tokens in your application program to meet your application requirements.

Table 10. Types of Parameter Lists

Variation	Guidelines	Reference
Single parameter list and a single token area	<ul style="list-style-type: none"> • Only one subcommand request can be active at a time. • An active subcommand request must be completed before beginning another subcommand request. 	“Coding a Single Parameter List, Single Token Area” on page 36
Single parameter list and multiple token area	<ul style="list-style-type: none"> • More than one subcommand request can be active at a time. • Only one subcommand request can be processed at any given time. 	“Coding a Single Parameter List, Multiple Token Areas” on page 38
Multiple parameter lists with a single token area	<ul style="list-style-type: none"> • Only one subcommand can be active at a time. • Different parameter lists can be used for these tasks: <ul style="list-style-type: none"> – Begin subcommand requests. – Continue subcommand requests. – Release resources. • Starting a new subcommand request ends any previous subcommand request. 	“Coding Multiple Parameter List, Single Token Area” on page 40.

Table 10. Types of Parameter Lists (continued)

Variation	Guidelines	Reference
Multiple parameter lists and multiple token area	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – Begin subcommand requests. – Continue subcommand requests. – Release resources. 	“Coding Multiple Parameter List, Multiple Token Areas” on page 40

For illustrative purposes, the examples use inline code segments with shortened code lines.

Coding a Single Parameter List, Single Token Area

Figure 10 on page 37 is an example of how your application program can use 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. You need a new token for the second 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)
...*****
** 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 10. Single Parameter List, Single Token Area

The example includes the OPERATION=RELEASE parameter. When you use OPERATION=RELEASE, DFSMSrmm releases work areas that contain data and pointers for the subcommand. 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.

For OPERATION=RELEASE, you do not specify SUBCMDADDR or OUTBUFADDR. For OPERATION=CONTINUE, you do not specify SUBCMDADDR.

Coding a Single Parameter List, Multiple Token Areas

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 11 shows how you can use 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 11. Single Parameter List, Multiple Token Areas (Part 1 of 2)

```

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

Figure 11 on page 38 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. When that token is reused to begin a new subcommand request, you cannot continue the previous request associated with that token.

In Figure 11 on page 38, 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 11 on page 38 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 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.

Coding Multiple Parameter List, Single Token Area

Figure 13 shows how you can use 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 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

Coding Multiple Parameter List, Multiple Token Areas

This variation lends itself to processing in re-entrant code where subroutines can be created for commonly used code. Figure 14 on page 41 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. Multiple Parameter Lists, Multiple Token Area (Part 1 of 2)

```

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

Specifying the Option to Free a Resource

You can free a resource when you no longer need to use it by performing one of these actions:

- Use the OPERATION=RELEASE and TOKEN=*token* parameters to free all resources associated with the specified token as shown in Figure 15 on page 43.

```

*****
** 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=OKENA

```

Figure 15. TOKEN= Specified on EDGXCI

Specifying TOKEN=OKENA on the EXECUTE form of EDGXCI causes the 4-byte OKENA 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.

- Specify the OPERATION=ENDALL parameter to free all resources associated with all tokens, as shown in Figure 16.

Rule: You are responsible for setting applicable tokens to all zeros when you specify OPERATION=ENDALL.

- Your application program ends (end-of-task occurs).

Specifying the Option to Release a Resource

To release a resource, you must have access to the tokens associated with the resources that 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 that resources can be freed:

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

In Figure 16, 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 16. TOKEN= Not Specified on EDGXCI

Chapter 5. Using an Alternative Interface to the DFSMSrmm Application Programming Interface

The EDGXHINT interface is an alternative interface to the DFSMSrmm application programming interface (API):

- Assembler or C/C++ programs can be linked together with module EDGXHINT to exploit the API interface provided.
- When using Java, you must use the Java Native Interface (JNI) to C/C++ before you can use EDGXHINT.

EDGXHINT is shipped as a load module in LINKLIB.

When using high level languages to write applications to obtain information about DFSMSrmm resources, you use the same DFSMSrmm subcommand strings that you can use with the EDGXCI interface. You get output as structured field introducers (SFIs). To receive output as an XML document, use the Object-Oriented DFSMSrmm Application Programming Interface Using C++.

Related Reading:

- *z/OS XL C/C++ User's Guide, SC09-4767*
- *Integrating Java with Existing Data and Applications on OS/390, SG24-5142-00*

To create a program exploiting the EDGXHINT interface, bind EDGXHINT together with your own module as shown in Figure 17.

```
//BINDPGM JOB (4378), 'BIND A PROGRAM',MSGCLASS=H,MSGLEVEL=(1,1),
//      TIME=3,CLASS=A,REGION=0M,NOTIFY=&SYSUID
//*
//*****
//*                                     ***
//* BIND A C/C++ PROGRAM TO USE THE EDGXHINT INTERFACE TO RMM          ***
//*                                     ***
//* SYSLMOD: OUTPUT DATASET (HLQ.CPP.LINKLIB) MUST BE PDSE FORMAT      ***
//*                                     ***
//*****
//BIND EXEC PGM=IEWL,REGION=4M,
//      PARM='AMODE=31,MAP,RENT'
//SYSLIB DD DSN=CEE.SCEELKEX,DISP=SHR
//      DD DSN=CEE.SCEELKED,DISP=SHR
//      DD DSN=CEE.SCEECPP,DISP=SHR
//SYSLMOD DD DISP=SHR,DSN=HLQ.CPP.LOAD
//SYSPRINT DD SYSOUT=*
//INOBJ DD DSN=HLQ.OBJ,DISP=SHR
//LINKLIB DD DISP=SHR,DSN=SYS2.LINKLIB
//SYSLIN DD *
//      INCLUDE INOBJ(USERPROG)
//      INCLUDE LINKLIB(EDGXHINT)
//      NAME USERPROG(R) RC=0
/*
```

Figure 17. Binding a C++ program for use of EDGXHINT

The application program must provide buffers for the:

- Command string you want to pass to the API

- Output you will receive back from the API. The minimum recommended size is 80KB. The larger the output buffer you provide, the more resources that can be returned by one call to EDGXHINT.
- Messages that may be issued by the API as result of your command. The minimum recommended size is 256 bytes

The application program also must fill an interface structure, which is used to communicate with the API. You can then call EDGXHINT by passing the pointer to the interface structure. For more details on the processing between your program and the RMM API, see Chapter 4, “Using the DFSMSrmm Application Programming Interface Using Assembler Language,” on page 33.

Parameter list to call EDGXHINT

Table 11. Parameter list for a call of EDGXHINT

Field	Description	Set from
Function code	<ol style="list-style-type: none"> 1. Open API (start communication) 2. Close API (end communication) 3. Issue command (begin a request) 4. Get next buffer (continue a request) 5. Release (end a request) 	User program
Pointer to the command buffer	The user program needs to obtain the storage for a buffer big enough to hold the TSO subcommand to be issued. Maximum is 255 byte. EDGXHINT will read the TSO command from this buffer.	User program
Pointer to the output buffer	The user program needs to obtain the storage for an output buffer. Minimum recommended is 80KB. EDGXHINT will use this buffer to return the data requested.	User program
Pointer to first message buffer	The user program must obtain the storage for a 256 byte buffer. This buffer should always be cleared before EDGXHINT is called, to delete pre-existing content. EDGXHINT will use this buffer to return a message resulting from the last issued command, if appropriate.	User program
Pointer to second message buffer	The user program must obtain the storage for a 256 byte buffer. This buffer should always be cleared before EDGXHINT is called, to delete pre-existing content. EDGXHINT will use this buffer to return a second message resulting from the last issued command, if appropriate.	User program
Message count	Number of messages returned by EDGXHINT	EDGXHINT
API address	Address of EDGXAPI. Set by OPEN function. Can be used to determine if the API is open. If not NULL, then API is open.	EDGXHINT
MTAB address	Address of the DFSMSrmm message table. Set by OPEN function, used by EDGXHINT internally.	EDGXHINT
CMSG address	Address of the DFSMSrmm message routine. Set by OPEN function, used by EDGXHINT internally.	EDGXHINT
Token	Token used by macro EDGXCI to identify the request. The token is created at BEGIN processing (function 3) and used by CONTINUE processing (function 4). The token is cleared (set to zero) by EDGXHINT after RELEASE processing (function 5).	EDGXHINT
Return code	API return code	EDGXHINT
Reason code	API reason code	EDGXHINT

Interface structure to pass the parameter list to EDGXHINT

In C/C++ programming language, a struct is used to pass the parameter list to EDGXHINT. Sample code for this purpose is shown in Figure 18. In this sample, the interface structure itself is defined in `t_interface`. Additional structs are used to map the command buffer (`t_comm`) and the output buffer (`t_outph`).

```
typedef struct t_comm          // to map the output buffer
{
    short com_length;          // length of the command
    char  commandBuffer[255]; // storage to hold the command
};
t_comm  commandStrct;        // variable of type t_comm
t_comm* commPtr;            // pointer to t_comm

typedef struct t_outph        // to map the command buffer header
{
    long  out_length;         // length of the output buffer
    long  out_needed;        // output buffer length needed
    long  out_used;          // output buffer length used
};
t_outph outputHeaderStrct;   // variable of type t_outph

typedef struct t_outp         // to map the output buffer
{
    t_outph header;          // output buffer header
    char  outputBuffer[80000]; // storage to hold the output
};
t_outp  outputStrct;        // variable of type t_outp
t_outp* outputPtr;         // pointer to t_outp

typedef struct t_interface    // to map the interface structure
{
    long    function;        // function code
    t_comm* command_ptr;     // pointer to command buffer
    t_outp* outputBuf_ptr;   // pointer to output buffer
    char*   messageBuf_ptr1; // pointer to first message buffer
    char*   messageBuf_ptr2; // pointer to second message buffer
    long    messageCount;    // number of messages returned
    void*   addr_XAPI;       // address of module EDGXAPI
    void*   addr_MTAB;       // address of module EDGMTAB
    void*   addr_CMSG;       // address of module EDGCMMSG
    long    token;           // token to identify the request
    long    returncode;      // API return code
    long    reasoncode;      // API reason code
};
t_interface interStrct;     // variable of type t_interface
t_interface* pI;           // pointer to t_interface
```

Figure 18. C/C++ sample code for an interface struct

Communication with the API

Define the API

Define the EDGXHINT program interface to your program, together with the interface struct, using code such as:

```
extern "C" int EDGXHINT( t_interface* );
```

Start API communication

To start API communication, first initialize all elements of the interface structure and clear the buffers you provide. You can then open a communication session with the API by setting the function code to 1 (=OPEN) and calling EDGXHINT, passing the pointer to the interface struct:

```
interStruct.function      = 1L;
EDGXHINT(pI);
```

You can use the return and reason code elements of the interface structure to determine whether the open process was successful:

```
if ( interStruct.returncode == 0L
    && interStruct.reasoncode == 0L )
    ....                // successfully opened the API session
else
    ....                // error handling needed
```

If the open process is successful, EDGXHINT fills the elements of the interface structure as described in Table 11 on page 46.

Issue a request

If the open is successful, you can start a request session by issuing a TSO subcommand through the API. Sample code for this is shown in Figure 19. Place the command string in the command buffer, initialize buffers, set function code to 3 (=BEGIN), and call EDGXHINT.

```
char command[12] = "SV OWNER(*)";           // define the command
strcpy(commandStrct.commandBuffer,command); // fill the command buffer
commandStrct.com_length = strlen(command)+2; // set the command length
                                           // command length + 2 byte length field
strcpy(outputStrct.outputBuffer,'\0');      // clear output buffer
outputStrct.header.out_used=0;
strcpy(interStruct.messageBuf_ptr1,'\0');   // clear message buffers
strcpy(interStruct.messageBuf_ptr2,'\0');
interStruct.function = 3L;                  // set function code
EDGXHINT(pI);                              // call EDGXHINT
```

Figure 19. Issue a TSO subcommand using EDGXHINT

You can evaluate the return and reason code to determine whether the command was processed successfully. From the message count, you can determine whether there are messages available in the message buffers. You will find returned data in the output buffer. This data is in SFI format and can be processed as described in Chapter 6, "Processing the Output Data in the Output Buffer," on page 51. If a search command was issued, you will find one or more complete resources in the output buffer.

EDGXHINT always uses the EDGXCI MULTI=YES keyword on behalf of its callers. Therefore, all callers must be updated, if necessary, to handle a buffer containing multiple resources. A caller requiring the return of just a single resource can use the LIMIT(1) operand on the SEARCH subcommand.

Continue a request

If more matching resources exist (returncode = 0, reasoncode = 4), you might want to continue the request session. Clear the buffers, set function code to 4 (=CONTINUE) and call EDGXHINT again. The next set of resources are returned to the output buffer.

End a request

To end the request session, release the corresponding token. Set function code to 5 (=RELEASE) and call EDGXHINT.

```
interStruct.function      = 5L;  
EDGXHINT(pI);
```

End API communication

To end communication with the API, set the function code to 2 (=CLOSE) and call EDGXHINT, passing the pointer to the interface struct:

```
interStruct.function      = 2L;  
EDGXHINT(pI);
```

Return and reason codes using EDGXHINT

When using interface EDGXHINT, you receive return and reason codes, as described in “EDGXCI Return and Reason Codes” on page 10.

Chapter 6. Processing the Output Data in the Output Buffer

The DFSMSrmm application programming interface returns data in the output buffer you define. The data is in this 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 that contains 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 110 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.

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.

The DFSMSrmm API does not return output data in the output buffer for every subcommand you issue using the API. See “SFIs for Output Data for Subcommands” on page 61 for information on each subcommand and the possible output data that the API returns as structured fields in your output buffer.

Description of Structured Fields

A structured field consists of:

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

Part	Description
------	-------------

SFI	Structured Field Introducer. A structure with a minimum size of 8 bytes in this 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 83 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.

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

Requesting SFI Data Format

You determine if the DFSMSrmm API returns line format or field format data to your application program. Line format contains fixed text and variable data that are formatted into lines. Line format is suitable for displaying at a terminal or for printing. Field format data 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

```
A00001: RMMUSER.TSO.COMMAND1.
```

is only one data set on the volume

Requesting Line Format

Figure 20 on page 53 is an example of the line format data that the DFSMSrmm API returns when you specify the OUTPUT=LINES parameter. In the example, the request specifies the RMM TSO subcommand LISTDATASET RMMUSER.TAPE VOLUME(A00001). The request might produce the output that is shown in Figure 20 on page 53. The value for <line> is the SFI for each line and is followed by the data returned from specifying the RMM LISTDATASET subcommand.


```

<Begin DATASET Group>
  <line>Data set name = RMMUSER.TAPE
  <line>Volume       = A06061           Physical file sequence number = 1
  <line>Owner        = RMMUSER           Data set sequence = 1
  <line>Create date  = 12/05/2007 Create time = 01:16:38 System ID       = EZU34
  <line>Expiration date = 12/10/2007 Original expir. date =
  <line>Block size   = 3120           Block count = 1
  <line>Data set size(KB) = 97656
  <line>Percent of volume = 0           Total block count = 1
  <line>Logical Record Length = 80       Record Format = FB
  <line>Date last written = 12/05/2007 Date last read = 12/05/2007
  <line>Job name       = RMMUSERJ       Last job name = RMMUSERJ
  <line>Step name      = WRITE          Last step name = WRITE
  <line>Program name   = IEBCGENER      Last program name = IEBCGENER
  <line>DD name        = SYSUT2         Last DD name = SYSUT2
  <line>Device number  = 0590          Last Device number = 0590
  <line>Management class =             VRS management value =
  <line>Storage group  =             VRS retention date =
  <line>Storage class  =             VRS retained = NO
  <line>Data class     =             Closed by Abend = NO
  <line>               =             Deleted = NO
  <line>               =             Catalog status = UNKNOWN
  <line>Primary VRS details:
  <line>  Name =
  <line>  Job name =           Type =
  <line>  Subchain NAME =       Subchain start date =
  <line>Secondary VRS details:
  <line>  Value or class =
  <line>  Job name =
  <line>  Subchain NAME =       Subchain start date =
  <line>Security Class =       Description =
  <line>
<End DATASET Group>

```

Figure 20. Example of List Type of Output Using OUTPUT=LINES

Requesting Field Format

Figure 21 on page 54 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 21 on page 54.

```

<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)        >
<TZ - Time Zone               : 4, bin(31)        >
<DEV - Device Number          : 4, character       >
<FILE - Physical File Sequence : 4, bin(31)        >
<CDTJ - Create Date           : 4, packed decimal  >
<CTM - Create Time            : 4, packed decimal  >
<SYS - Creating 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/Referenced: 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 Job Name  : 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 - Security Class Descriptio: 32, character      >
<ABND - Closed by Abend      : 1, bin(8)        >
<CTLG - Catalog status       : 1, bin(8)        >
<2JBN - Secondary VRS jobname mas: 8, character       >
<2NME - Secondary VRS mask    : 8, character       >
<2SCN - Secondary VRS subchain na: 8, character       >
<2SCD - Secondary VRS subchain da: 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)        >
<XDTJ - Expiration Date      : 4, packed decimal  >
<OXDJ - Original Expiration Date : 4, packed decimal  >
| <DLTD - Deleted By Disposition Pr: 1, bin(8)        >
| <DSS6 - Data Set Size        : 14, compound         >
<End DATASET Group>

```

Figure 21. Example of Output Using *OUTPUT=FIELDS*

Figure 21:

- Shows begin and end group SFIs. In this example, <Begin DATASET Group> and <End DATASET Group>.
- Includes descriptive names used to identify SFIs. 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 37 on page 69 occurs before OWN for LISTDATASET while OWN occurs before VOL for LISTPRODUCT in Figure 39 on page 70.
- 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 115 for an example of an output buffer in hexadecimal representation.

Requesting 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 6.

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 22.

```
<Begin DATASET Group>
<DSN - Data Set Name      : 44, character      >RMMUSER.DATA01
<VOL - Volume Serial      : 6, character       >V10000
<OWN - Owner              : 8, character       >RMMUSER
<TZ - Time Zone          : 4, bin(31)        >x'FFFF9D90'
<CDTJ - Create Date      : 4, packed decimal >x'2007339F'
<CTM - Create Time       : 4, packed decimal >x'0116362F'
<FILE - Physical File Sequence : 4, bin(31)    >x'00000001'
<XDTJ - Expiration Date  : 4, packed decimal >x'2007344F'
<End DATASET Group>
```

Figure 22. Example of Search Type of Output Using EXPAND=NO

Refer to Appendix D, “Hexadecimal Example of an Output Buffer,” on page 115 for a hexadecimal representation and discussion of the contents of the output buffer shown in Figure 22.

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 23.

```

<Begin DATASET Group>
<DSN - Data Set Name           : 44, character           >RMMUSER.TAPE
<CJBN - Job Name               : 8, character            >RMMUSERJ
<VOL - Volume Serial          : 6, character            >A06061
<OWN - Owner                   : 8, character            >RMMUSER
<DSEQ - Data Set Sequence     : 4, bin(31)             >x'00000001'
<TZ - Time Zone               : 4, bin(31)             >x'FFFF9D90'
<DEV - Device Number          : 4, character            >0590
<FILE - Physical File Sequence : 4, bin(31)             >x'00000001'
<CDTJ - Create Date           : 4, packed decimal       >x'2007339F'
<CTM - Create Time            : 4, packed decimal       >x'0116381F'
<SYS - Creating system ID     : 8, character            >EZU0000
<BLKS - Block Size            : 4, bin(31)             >x'00000C30'
<BLKC - Block Count           : 4, bin(31)             >x'00000001'
<LRCL - Logical Record Length : 4, bin(31)             >x'00000050'
<RCFM - Record Format         : 4, character            >FB
<DC - Data Class              : 8, character            >
<DLWJ - Date Last Written     : 4, packed decimal       >x'2007339F'
<DLRJ - Date Last Read/Referenced: 4, packed decimal >x'2007339F'
<STEP - Step Name            : 8, character            >WRITE
<DD - DD Name                 : 8, character            >SYSUT2
<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)             >x'00'
<VJBN - Primary VRS Job Name  : 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)             >x'00'
<NME - Security Class Name   : 8, character            >
<CLS - Security Class Descriptio: 32, character         >
<ABND - Closed by Abend      : 1, bin(8)             >x'00'
<CTLG - Catalog status      : 1, bin(8)             >x'00'
<2JBN - Secondary VRS jobname mas: 8, character         >
<2NME - Secondary VRS mask   : 8, character            >
<2SCN - Secondary VRS subchain na: 8, character         >
<2SCD - Secondary VRS subchain da: 4, packed decimal     >
<BLKT - Total block count    : 4, bin(31)             >x'00000001'
<CPGM - Creating program name : 8, character            >IEBGENER
<LPGM - Last used program name : 8, character            >IEBGENER
<LJOB - Last used job        : 8, character            >RMMUSERJ
<LSTP - Last used step name   : 8, character            >WRITE
<LDD - Last used DD name     : 8, character            >SYSUT2
<LDEV - Last Drive           : 4, character            >0590
<DPCT - Percent of volume    : 1, bin(8)             >x'00'
<XDTJ - Expiration Date     : 4, packed decimal       >x'2007344F'
<OXDJ - Original Expiration Date : 4, packed decimal >
| <DLTD - Deleted By Disposition Pr: 1, bin(8)           >x'00'
| <DSS6 - Data Set Size       : 14, compound             >x'010303010A0600000000000017D78'
<End DATASET Group>

```

Figure 23. Example of Search Type of Output Using OUTPUT=FIELDS, EXPAND=YES

Accessing 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 *z/OS DFSMSrmm Managing and Using Removable Media* 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 10 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 59 for more information.

Accessing 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 60 for information about which messages can be placed in your output buffer.

Interpreting 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 98cccc 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: *hhmsstC*, where *hhmsst* is the time in hours, minutes, seconds, and tenths of seconds and *C* is a standard packed-decimal sign character.

Using Different Time Zones

Default dates and times are returned in the time zone of the DFSMSrmm system processing the subcommand. The TZ SFI provides the time zone offset so if necessary, the application can convert dates and times to any other required time zone. When issuing subcommands that specify date or time values, such as ADDDATASET or CHANGEVOLUME, you can specify the TZ operand to indicate to the DFSMSrmm system the time zone offset the application is using. DFSMSrmm converts dates and times to UTC/GMT/local time in order to store them in the DFSMSrmm control data set. Refer to *z/OS DFSMSrmm Implementation and Customization Guide* for more information on creating or updating the DFSMSrmm control data set control record and setting up DFSMSrmm common time support.

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 59
 - “Messages and Message Variables SFIs” on page 60
 - “ADD-Type of Subcommands” on page 62
 - “CHANGE-Type of Subcommands” on page 62
 - “DELETE-Type of Subcommands” on page 63
 - “GETVOLUME Subcommand” on page 63
 - “LIST-Type of Subcommands” on page 63
 - “SEARCH-Type of Subcommands” on page 74

This notation indicates an SFI:

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

where “xxx” is a character type of mnemonic. In your application program, you need to use the 3-byte or 4-byte hexadecimal identifiers for Structured Field Introducers.

Appendix A, “Structured Field Introducers,” on page 83 describes all the structured fields that the DFSMSrmm API can return to your application program.

Appendix B, “Structured Field Introducers by Subcommand,” on page 105 shows all of the Structured Field Introducers by subcommand.

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 on page 59.

```

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

```

Figure 25. Begin and End Resource Group SFI Sequence

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.

```

<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 OPENRULE group> <End OPENRULE group>
<Begin OWNER group>   <End OWNER group>
<Begin PRODUCT group> <End PRODUCT group>
<Begin PRTITION group> <End PRTITION 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

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, SEARCHVOLUME, LISTPRODUCT, and SEARCHPRODUCT 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 MEDINF group> <End MEDINF group>
<Begin MNTMSG group> <End MNTMSG group>
<Begin MOVES group>  <End MOVES group>
<Begin OPTION group> <End OPTION group>
<Begin PRODVOL group> <End PRODVOL 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

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 on page 60. 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

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 use the CONTINUE operand on any SEARCH subcommand, the DFSMSrmm API returns the continue information at the message group with the CONT SFI as shown in Figure 30.

```

<Begin VOLUME group>
  <Begin MESSAGE group>
    <MSGN - message number      : 8 , character:      >
    <ENTN - number of entries    : 4 , bin(1):         >
  <End MESSAGE group>
  <Begin MESSAGE group>
    <MSGN - message number      : 8 , character:      >
    <CONT -continue information  :84 , character:      >
  <End MESSAGE group>
<End VOLUME group>

```

Figure 30. Message Group with the CONT SFI

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 12 on page 61 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 was successful.

Table 12 on page 61 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 12. 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
EDG3019	RCK	DB DR	12	many
EDG3020	CNT	DB DR	12	many
EDG3025	CONT	All SEARCH subcommands	4	2
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

For a detailed explanation of these messages, see *z/OS MVS System Messages, Vol 5 (EDG-GFS)*. For a description of messages, use LookAt, described in “Using LookAt to look up message explanations” on page x. For DFSMSrmm return and reason codes, see *z/OS DFSMSrmm Managing and Using Removable Media*.

SFIs for Output Data for Subcommands

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 31 on page 62. DFSMSrmm places system return codes and reason codes in your output buffer as described in “System Return and Reason Code SFIs” on page 59.

```

<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 31. Formatted Lines

When you specify OUTPUT=FIELDS, the DFSMSrmm API returns output data as unformatted data in structured fields.

ADD-Type of Subcommands

The DFSMSrmm ADD-type of subcommands are: ADDDBIN, 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 under these conditions:

- You specify the ADDVOLUME subcommand with the POOL operand. The DFSMSrmm API returns the rack number that is assigned to the volume in the format as shown in Figure 32.
- An error occurs for specific return and reason code combinations described in “Messages and Message Variables SFIs” on page 60 and “SFIs for Return and Reason Codes” on page 85.

```

<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 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 in the format as shown in Figure 33.
- When an error occurs for specific return and reason code combinations described in “Messages and Message Variables SFIs” on page 60 and “SFIs for Return and Reason Codes” on page 85.

```

<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 33. SFIs for CHANGEVOLUME with OUTPUT=FIELDS

DELETE-Type of Subcommands

The DFSMSrmm DELETE-type of subcommands are: DELETEBIN, DELETEDATASET, DELETEOWNER, DELETEPRODUCT, DELETETRACK, 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 for specific return and reason code combinations described in “Messages and Message Variables SFIs” on page 60 and “SFIs for Return and Reason Codes” on page 85.

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 34.
- When an error occurs, and then only for specific return and reason code combinations described in “Messages and Message Variables SFIs” on page 60 and “SFIs for Return and Reason Codes” on page 85.

```
<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 34. 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 SFIs”
- “LISTCONTROL SFIs” on page 64
- “LISTDATASET SFIs” on page 68
- “LISTOWNER SFIs” on page 69
- “LISTPRODUCT SFIs” on page 70
- “LISTRACK SFIs” on page 70
- “LISTVOLUME SFIs” on page 71
- “LISTVRS SFIs” on page 73

LISTBIN SFIs

The SFIs produced for the LISTBIN subcommand with OUTPUT=FIELDS are shown in Figure 35 on page 64.

```

<Begin RACK/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  >
  <MIV - Moving-In Volume        : 6, character  >
  <MOV - Moving-Out Volume       : 6, character  >
  <OVOL - Old Volume             : 6, character  >
  <TZ  - Time Zone                : 4, bin(31)   >
<End RACK/BIN Group>

```

Figure 35. SFIs for LISTBIN with OUTPUT=FIELDS

LISTCONTROL SFIs

The SFIs produced for the LISTCONTROL subcommand with OUTPUT=FIELDS are shown in Figure 36 on page 65.

```

| <Begin CONTROL Group>
|   <Begin CNTL Group>
|     <TZ   - Time Zone           : 4, bin(31)       >
|     <MTP  - CDS type            : 1, bin(8)        >
|     <MDTJ - CDS Create Date     : 4, packed decimal >
|     <MTM  - CDS Create Time     : 4, packed decimal >
|     <UDTJ - CDS Last update date : 4, packed decimal >
|     <UTM  - CDS Last update time : 4, packed decimal >
|     <JRNU - Journal Percentage Used : 2, bin(15)    >
|     <JRNF - JOURNALFULL Parmlib Value: 2, bin(15)    >
|     <JRNS - Journal status      : 1, bin(8)        >
|     <BDTJ - Last CDS Backup Date : 4, packed decimal >
|     <BTM  - Last CDS Backup Time : 4, packed decimal >
|     <JBDM - Last journal backup date : 4, packed decimal >
|     <JBDM - Last journal backup time : 4, packed decimal >
|     <XDTJ - Expiration Date     : 4, packed decimal >
|     <XTM  - Last Inven Mgt Expir Time: 4, packed decimal >
|     <RDTJ - Last CDS Extract Date : 4, packed decimal >
|     <RTM  - Last CDS Extract Time : 4, packed decimal >
|     <DDTJ - Delete/Store Date   : 4, packed decimal >
|     <DTM  - Last Store update run tim: 4, packed decimal >
|     <SOSJ - Last XPROC Start Date : 4, packed decimal >
|     <SOST - Last XPROC Start Time : 4, packed decimal >
|     <VDTJ - Last Inven Mgt Proc Date : 4, packed decimal >
|     <VTM  - Last Inven Mgt VRS Time : 4, packed decimal >
|     <LRK  - # Library Rack Numbers : 4, bin(31)    >
|     <FRK  - Free Rack Num in Library : 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 Numbers in REMOT: 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 Synch 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)     >
|     <X300 - EDGUX300 exit status   : 1, bin(8)     >
|     <EBIN - Extended Bin Status    : 1, bin(8)     >
|     <CDSU - CDS percentage used     : 2, bin(15)    >
|     <CSHN - Client/Server host name : 63, character >
|     <CSIP - Client/Server IP address : 15, character >
|     <UTC  - Common time status enable: 1, bin(8)     >
|     <CDSQ - CDS id ENQ name enabled : 1, bin(8)     >
|   <End CNTL Group>
|   <Begin OPTION Group>
|     <OPM  - Operating Mode         : 1, bin(8)      >
|     <DRP  - Default Retention Period : 4, bin(31)   >
|     <MRP  - Maximum Retention Period : 4, bin(31)   >

```

Figure 36. SFIs for LISTCONTROL with OUTPUT=FIELDS (Part 1 of 4)

```

| <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 Required in IP: 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 Record Numbe: 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 - SMF System ID : 8, character >
| <BLP - BLP Option : 1, bin(8) >
| <NOT - Notify : 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 - VRSCCHANGE Value : 1, bin(8) >
| <VRSL - VRSEL Value : 1, bin(8) >
| <PSFX - Parmlib Member Suffix : 2, character >
| <PSF2 - Parmlib Member Suffix 2 : 2, character >
| <VACT - VRSMIN action : 1, bin(8) >
| <VMIN - VRSMIN Count Value : 4, bin(31) >
| <JRNT - Journal transaction : 1, bin(8) >
| <VDRA - VRS Drop Action : 1, bin(8) >
| <VDRC - VRS Drop Count : 4, bin(31) >
| <VDRP - VRS Drop Percentage : 2, bin(15) >
| <VREA - VRS Retain Action : 1, bin(8) >
| <VREC - VRS Retain Count : 4, bin(31) >
| <VREP - VRS Retain Percentage : 2, bin(15) >
| <XDRA - EXPDT Drop Action : 1, bin(8) >
| <XDRC - EXPDT Drop Count : 4, bin(31) >
| <XDRP - EXPDT Drop Percentage : 2, bin(15) >
| <DSPD - Disposition DD name : 8, character >
| <DSPM - Disposition message pref: 8, character >
| <RTBY - Retain by : 1, bin(8) >
| <MVBY - Move by : 1, bin(8) >
| <GDGC - GDG cycleby : 1, bin(8) >
| <GDGD - GDG duplicate : 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, bin(8) >
| <SMP - System managed tape purge: 1, bin(8) >
| <SMU - System managed tape updat: 1, bit(8) >
| <ACS - SMS ACS support : 1, bin(8) >
| <PACS - Pre-ACS support : 1, bin(8) >
| <RUB - Reuse Bin at : 1, bin(8) >
| <CMDD - Command Auth DSN : 1, bin(8) >
| <CMDO - Command Auth Owner : 1, bin(8) >
| <MEDN - Media Name : 8, character >
| <LCTK - Local Task : 4, bin(31) >
| <SSTY - Subsystem type : 1, bin(8) >
| <SRHN - Server host name : 63, character >
| <SRIP - Server IP Address : 15, character >
| <SRPN - Server port number : 4, bin(31) >
| <SRTK - Server task : 4, bin(31) >
| <End OPTION Group>

```

Figure 36. SFIs for LISTCONTROL with OUTPUT=FIELDS (Part 2 of 4)

```

| <Begin SECCLS Group>
|   <SEC - Security Class Number   : 1, bin(8)           >
|   <NME - Security Class Name     : 8, character        >
|   <SCST - Security Class Status  : 1, bit(8)           >
|   <CLS - Security Class Descriptio: 32, character       >
| <End SECCLS Group>
| <Begin VLPOOL Group>
|   <PID - Pool Prefix             : 6, character        >
|   <PSN - Pool Definition System ID: 8, character      >
|   <PRF - Pool Def RACF Option    : 1, bin(8)         >
|   <PTP - Pool Def Pool Type     : 1, bin(8)         >
|   <XDC - Expiration Date Check  : 1, bin(8)         >
|   <ACT - Action on Release       : 1, bit(8)         >
|   <SCRM - Scratch mode          : 1, bin(8)         >
|   <PLN - Pool Name               : 8, character      >
|   <MEDN - Media Name             : 8, character      >
|   <PDS - Pool Description        : 40, character     >
|   <MOP - Master Overwrite       : 1, bin(8)         >
| <End VLPOOL Group>
| <Begin MNTMSG Group>
|   <MID - Mount Message ID       : 12, character      >
|   <SMI - Offset to Message ID   : 2, bin(15)        >
|   <OVL - Offset to Volume Serial : 2, bin(15)        >
|   <OPL - Offset Rack Num or Pool I: 2, bin(15)      >
| <End MNTMSG Group>
| <Begin REJECT Group>
|   <GRK - Generic Rack Number    : 6, character      >
|   <TAC - Reject Type            : 1, bin(8)         >
| <End REJECT Group>
| <Begin LOCDEF Group>
|   <LDDF - Location Definition Exist: 1, bin(8)       >
|   <LDLC - Location Name         : 8, character      >
|   <LDMT - Location Management 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 MEDINF Group>
|   <MDNF - Media Information Name : 8, character      >
|   <MEDT - Media Type            : 1, bin(8)         >
|   <MDTX - External Media Type   : 8, character      >
|   <MEDR - Media Recording Format  : 1, bin(8)         >
|   <MDRX - External Recording Techn.: 8, character    >
|   <VCAP - Volume capacity       : 4, bin(31)        >
|   <MDRP - MEDINF Replace Policy Per: 4, bin(31)      >
|   <MDRT - MEDINF Replace Policy Tem: 4, bin(31)      >
|   <MDRW - MEDINF Replace Policy Wri: 4, bin(31)      >
|   <MDRA - MEDINF Replace Policy Age: 4, bin(31)      >
| <End MEDINF Group>
| <Begin PRTITION Group>
|   <PTVL - Volume Serial Number  : 6, character      >
|   <PTVS - Volume Range Start    : 6, character      >
|   <PTVE - Volume Range End     : 6, character      >
|   <PTTP - Type of Partition Entry : 1, bin(8)         >
|   <PTSA - SMT Action for Partition: 1, bin(8)         >
|   <PTNA - NOSMT Action for Partitio: 1, bin(8)         >
|   <PTNL - Location Name        : 8, character      >
| <End PRTITION Group>

```

Figure 36. SFIs for LISTCONTROL with OUTPUT=FIELDS (Part 3 of 4)

```

| <Begin OPENRULE Group>
|   <ORVL - Volume Serial Number      : 6, character      >
|   <ORVS - Volume Range Start        : 6, character      >
|   <ORVE - Volume Range End          : 6, character      >
|   <ORTP - Type of Openrule Entry    : 1, bin(8)       >
|   <ORIA - Input Action               : 1, bin(8)       >
|   <ORII - Input Ignore Condition     : 1, bit(8)       >
|   <ORIR - Input Reject Condition     : 1, bit(8)       >
|   <OROA - Output Action              : 1, bin(8)       >
|   <OROI - Output Ignore Condition    : 1, bit(8)       >
|   <OROR - Output Reject Condition    : 1, bit(8)       >
| <End OPENRULE Group>
| <Begin ACTIONS Group>
|   <ACT - Action 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 36. SFIs for LISTCONTROL with OUTPUT=FIELDS (Part 4 of 4)

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 no data.

When DFSMSrmm cannot return all the output data for the LISTCONTROL subcommands in your output buffer, you must specify OPERATION=CONTINUE after processing your output buffer to obtain the rest of the LISTCONTROL output data.

Related Reading: See “Using the CONTINUE Operation in the EDGXCI Macro” on page 33 for additional information.

LISTDATASET SFIs

The SFIs produced for the LISTDATASET subcommand with OUTPUT=FIELDS are shown in Figure 37 on page 69.


```

<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)       >
<TZ - Time Zone                 : 4, bin(31)       >
<DEV - Device Number           : 4, character      >
<FILE - Physical File Sequence : 4, bin(31)       >
<CDTJ - Create Date            : 4, packed decimal >
<CTM - Create Time             : 4, packed decimal >
<SYS - Creating 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/Referenced: 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 Job Name   : 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 - Security Class Descriptio: 32, character     >
<ABND - Closed by Abend        : 1, bin(8)       >
<CTLG - Catalog status         : 1, bin(8)       >
<2JBN - Secondary VRS jobname mas: 8, character      >
<2NME - Secondary VRS mask     : 8, character      >
<2SCN - Secondary VRS subchain na: 8, character      >
<2SCD - Secondary VRS subchain da: 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)       >
<XDTJ - Expiration Date        : 4, packed decimal >
<OXDJ - Original Expiration Date : 4, packed decimal >
| <DLTD - Deleted By Disposition Pr: 1, bin(8)       >
| <DSS6 - Data Set Size         : 14, compound       ><End DATASET Group>

```

Figure 37. SFIs for LISTDATASET with OUTPUT=FIELDS

LISTOWNER SFIs

The SFIs produced for the LISTOWNER subcommand with OUTPUT=FIELDS are shown in Figure 38 on page 70.

```

<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   >
  <ADL1 - Address Line 1      : 40, character   >
  <ADL2 - Address Line 2      : 40, character   >
  <ADL3 - Address Line 3      : 40, character   >
  <ITL - Owner's Internal Tele Num: 8, character   >
  <ETL - Owner's Ext Telephone Num: 20, character  >
  <EMU - Owner's User ID      : 8, character    >
  <EMN - Owner's Node         : 8, character    >
  <VLN - Number of Volumes    : 4, bin(31)    >
  <EML - Owner's Email Address : 63, character  >
  <TZ  - Time Zone           : 4, bin(31)    >
<End OWNER Group>

```

Figure 38. SFIs for LISTOWNER with OUTPUT=FIELDS

LISTPRODUCT SFIs

The SFIs produced for the LISTPRODUCT subcommand with OUTPUT=FIELDS are shown in Figure 39.

```

<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   >
  <VLN  - Number of Volumes        : 4, bin(31)    >
  <TZ   - Time Zone                : 4, bin(31)    >
  <Begin PRODVOL Group>
    <VOL  - Volume Serial          : 6, character    >
    <RCK  - Rack or Bin Number     : 6, character    >
    <FCD  - Product Feature Code   : 4, character    >
  <End PRODVOL Group>
<End PRODUCT Group>

```

Figure 39. SFIs for LISTPRODUCT with OUTPUT=FIELDS

The PRODVOL group is repeated for each product volume.

LISTRACK SFIs

The SFIs produced for the LISTRACK subcommand with OUTPUT=FIELDS are shown in Figure 40.

```

<Begin RACK/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    >
  <PID - Pool Prefix             : 6, character    >
  <TZ  - Time Zone               : 4, bin(31)    >
<End RACK/BIN Group>

```

Figure 40. SFIs for LISTRACK with OUTPUT=FIELDS

LISTVOLUME SFIs

The SFIs produced for the LISTVOLUME subcommand with OUTPUT=FIELDS are shown in Figure 41 on page 72.

```

<Begin VOLUME Group>
  <Begin VOL Group>
    <VOL - Volume Serial          : 6, character      >
    <RCK - Rack or Bin Number    : 6, character      >
    <OWN - Owner                  : 8, character      >
    <TZ - Time Zone               : 4, bin(31)         >
    <CJBN - Job Name              : 8, character      >
    <CDTJ - Create Date           : 4, packed decimal >
    <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 at : 1, bin(8)         >
    <AVL - Volume Availability     : 1, bit(8)         >
    <LBL - Volume Label           : 1, bit(8)         >
    <DEN - Media Density           : 1, bin(8)         >
    <MDNF - Media Information Name  : 8, character     >
    <MEDT - Media Type             : 1, bin(8)         >
    <MDTX - External Media Type    : 8, character     >
    <MEDR - Media Recording Format  : 1, bin(8)         >
    <MDRX - External Recording Techn.: 8, character     >
    <MEDC - Media Compaction       : 1, bin(8)         >
    <MEDA - Media Special Attributes : 1, bin(8)         >
    <ACT - Action 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 or VRS Description: 30, character    >
    <NME - Security Class Name     : 8, character     >
    <CLS - Security Class Descriptio: 32, character    >
    <VRSI - Scratch Immediate      : 1, bin(8)         >
    <VRXI - Expiration date ignore : 1, bin(8)         >
    <VOLT - Volume Type            : 1, bin(8)         >
    <LVC - Current label version   : 1, bin(8)         >
    <LVN - Required label version  : 1, bin(8)         >
    <RBYS - Retain by set         : 1, bin(8)         >
    <STVC - Stacked volume count   : 4, bin(31)         >
    <SYS - Creating system ID      : 8, character     >
    <DSYS - Creation System IDfirst f: 8, character     >
    <VOL1 - VOL1 label volser     : 6, character     >
    <WWID - Worldwide ID          : 24, character    >
    <VNDR - Vendor                 : 8, character     >
    <KEL1 - Encryption Key Label 1 : 64, character    >
    <KEL2 - Encryption Key Label 2 : 64, character    >
    <KEM1 - Encryption Encoding mech : 5, character     >
    <KEM2 - Encryption Encoding mech : 5, character     >
    <WORM - WORM flag             : 1, bin(8)         >
  <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)         >
    <IRMM - IRMM Use              : 1, bin(8)         >
    <UID01- User ID 1             : 8, character     >
  <End ACCESS Group>

```

Figure 41. SFIs for LISTVOLUME with OUTPUT=FIELDS (Part 1 of 2)

```

<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/Referenced: 4, packed decimal >
  <DLWJ - Date Last Written  : 4, packed decimal >
  <LDEV - Last Drive         : 4, character     >
  <SEQ - Volume Sequence     : 4, bin(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) >
  <VWMC - Volume Write Mount Count : 4, bin(31) >
  <USE6 - Volume Usage      : 14, compound    >
<End STAT Group>
<Begin STORE Group>
  <LOC - Location           : 8, character     >
  <LOCT - Location Type    : 1, bin(8)       >
  <DEST - Destination Name  : 8, character     >
  <DSTT - Destination Type : 1, bin(8)       >
  <INTR - Volume Intransit Status : 1, bin(8) >
  <HLOC - Home Location     : 8, character     >
  <HLOT - Home Location Type : 1, bin(8) >
  <OLOC - Old Location      : 8, character     >
  <OLOT - Old Location Type : 1, bin(8) >
  <NLOC - Required Location : 8, character     >
  <NLOT - Required Location Type : 1, bin(8) >
  <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    >
  <DBIN - Destination Bin number : 6, character >
  <DBMN - Destination Bin media nam: 8, character >
<End STORE Group>
<End VOLUME Group>

```

Figure 41. SFIs for LISTVOLUME with OUTPUT=FIELDS (Part 2 of 2)

LISTVRS SFIs

The SFIs produced for the LISTVRS subcommand with OUTPUT=FIELDS are shown in Figure 42 on page 74.

```

<Begin VRS Group>
<VRS - Vital Record Specificatio: 44, character      >
<TYP - VRS Type                    : 1, bit(8)      >
<VJBN - Primary VRS Job Name       : 8, character    >
<VRC - Vital Record Count         : 4, bin(31)      >
<RET - Retention Type              : 3, bin(8)      >
<VDD - VRS Delay Days              : 2, bin(15)     >
<LOC - Location                    : 8, character    >
<SCI - 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    >
<TZ - Time Zone                    : 4, bin(31)     >
<DDTJ - Delete/Store Date          : 4, packed decimal >
<VANX - Next VRS Type              : 1, bin(8)      >
<VRSI - Scratch Immediate          : 1, bin(8)      >
<VRXI - Expiration date ignore    : 1, bin(8)      >
<DLRJ - Date Last Read/Referenced: 4, packed decimal >
<TLR - Time Last Read/Referenced: 4, packed decimal >
<End VRS Group>

```

Figure 42. SFIs for LISTVRS with OUTPUT=FIELDS

SEARCH-Type of Subcommands

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

| When you specify OUTPUT=FIELDS, the DFSMSrmm API returns data for all
 | SEARCH type of subcommands as structured fields. DFSMSrmm returns the
 | output data for one or more resources in your output buffer each time you call the
 | API. Use the MULTI=YES keyword to specify that your application can handle
 | multiple resources returned in your output buffer. You must specify
 | OPERATION=CONTINUE after processing your output buffer to obtain the output
 | data for the next resource or set of resources. Continue to call the DFSMSrmm API
 | until the output data for all matching resources has been returned.

Related Reading: See “Using the CONTINUE Operation in the EDGXCI Macro” on page 33 for additional information.

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 SFIs

Figure 43 on page 75 shows the output that DFSMSrmm returns when you specify the SEARCHBIN subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters.

```

<Begin RACK/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      >
  <MIV - Moving-In Volume       : 6, character      >
  <MOV - Moving-Out Volume       : 6, character      >
  <OVOL - Old Volume            : 6, character      >
  <TZ - Time Zone                : 4, bin(31)         >
<End RACK/BIN Group>

```

Figure 43. SFIs for SEARCHBIN with OUTPUT=FIELDS,EXPAND=NO

SEARCHDATASET SFIs

Figure 44 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 SFIs" on page 68 for LISTDATASET.

```

<Begin DATASET Group>
  <DSN - Data Set Name           : 44, character      >
  <VOL - Volume Serial           : 6, character      >
  <OWN - Owner                   : 8, character      >
  <TZ - Time Zone                : 4, bin(31)         >
  <CDTJ - Create Date            : 4, packed decimal >
  <CTM - Create Time             : 4, packed decimal >
  <FILE - Physical File Sequence : 4, bin(31)         >
  <XDTJ - Expiration Date        : 4, packed decimal >
<End DATASET Group>

```

Figure 44. SFIs for SEARCHDATASET with OUTPUT=FIELDS,EXPAND=NO

SEARCHOWNER SFIs

Figure 45 shows the output DFSMSrmm returns when you specify the SEARCHOWNER subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters.

```

| <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):        >
| <EML - owner's email address   : 63 , character:      >
| <TZ - time zone                : 4 , bin(31):        >
| <End OWNER group>
|

```

Figure 45. SFIs for SEARCHOWNER with OUTPUT=FIELDS,EXPAND=NO

SEARCHPRODUCT SFIs

Figure 46 shows the output DFSMSrmm returns when you specify the SEARCHPRODUCT subcommand and the EDGXCI macro OUTPUT=FIELDS parameter.

EXPAND=NO and EXPAND=YES return the same data elements so the EXPAND parameter can be omitted. Unlike LISTPRODUCT the SEARCHPRODUCT command returns only the PRODVOL group for the first product volume, if at least one volume exists.

```
<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 >
<VLN - Number of Volumes : 4, bin(31) >
<TZ - Time Zone : 4, bin(31) >
<Begin PRODVOL Group>
<VOL - Volume Serial : 6, character >
<RCK - Rack or Bin Number : 6, character >
<FCD - Product Feature Code : 4, character >
<End PRODVOL Group>
<End PRODUCT Group>
```

Figure 46. SFIs for SEARCHPRODUCT with OUTPUT=FIELDS

SEARCHRACK SFIs

Figure 47 shows the output DFSMSrmm returns when you specify the SEARCHRACK subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters.

```
<Begin RACK/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 >
<PID - Pool Prefix : 6, character >
<TZ - Time Zone : 4, bin(31) >
<End RACK/BIN Group>
```

Figure 47. SFIs for SEARCHRACK with OUTPUT=FIELDS,EXPAND=NO

SEARCHVOLUME SFIs

Figure 48 on page 77 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 SFIs" on page 71 for LISTVOLUME.


```

<Begin VOLUME Group>
  <VOL - Volume Serial          : 6, character      >
  <OWN - Owner                  : 8, character      >
  <RCK - Rack or Bin Number    : 6, character      >
  <TZ - Time Zone              : 4, bin(31)        >
  <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 Name     : 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, bin(8)         >
  <LVN - Required label version : 1, bin(8)         >
<End VOLUME Group>

```

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

SEARCHVRS SFIs

Figure 49 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 SFIs” on page 73 for LISTVRS.

```

<Begin VRS Group>
  <VRS - Vital Record Specificatio: 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      >
  <TZ - Time Zone              : 4, bin(31)        >
  <DDTJ - Delete/Store Date     : 4, packed decimal  >
  <VANX - Next VRS Type        : 1, bin(8)         >
  <VRSI - Scratch Immediate     : 1, bin(8)         >
  <VRXI - Expiration date ignore : 1, bin(8)         >
  <VRC - Vital Record Count     : 4, bin(31)        >
  <SC1 - Store Number          : 4, bin(31)        >
<End VRS Group>

```

Figure 49. 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.
- Whether you want one or multiple resources in your output buffer
- 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:

- 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 that you specify, the search criteria that you specify (fully or partially qualified names), and whether you specify a LIMIT value or LIMIT(*), DFSMSrmm can also issue these return codes and reason codes.

- Return code 0, reason code 0.
- Return code 4, reason code 8.

For more information about the return codes and reason codes that the API returns, see Table 9 on page 34.

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

Figure 50, Figure 51 on page 80, and Figure 52 on page 80 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 50:

- 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 when there is more output data. Specify the EDGXCI macro OPERATION=CONTINUE parameter to continue the subcommand request..

```
<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 50. CONTINUE Example, First Output Buffer

Second Output Buffer

After processing 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 51. CONTINUE Example, Second Output Buffer

In Figure 51:

- 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 these statements 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      Medianame  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 52. CONTINUE Example, Third (Last) Output Buffer

In Figure 52:

- The first two lines after the Begin RACK group are the header lines that you saw in the first output buffer. This is the output for a second search that the DFSMSrmm API started when you specified OPERATION=CONTINUE in response to the return code 4 and reason code 2.
- 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 this 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 <ul style="list-style-type: none">• 0 = 8-byte, fixed-length SFI
6	1-byte (Reserved)
7	1-byte data type: Type of data, if any, that follows the SFI <ul style="list-style-type: none">• 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(32) (4-byte (hex) unsigned value)• 6=Binary(64) (8-byte (hex) value)• 7=Character (variable-length)• 8=Compound SFI (multiple related values, see “Compound SFI” on page 84.)• 9=(4 bytes) Packed decimal Julian date: yyyydddC• A=(4 bytes) Packed decimal time format: hhmmssC

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).

Compound SFI

A compound SFI includes multiple values each with own data type and length.

Compound type:

- 1 Factored. A Binary(8) value combined with a second field containing a count. The second field is identified by a data type.

Factor values:

- 0 Bytes (unfactored)
- 1 KB
- 2 MB
- 3 GB
- 4 TB

and so on.

Compound SFIs follow this structure;

Byte Count	Description
8	Standard SFI including 1 byte data type identifier (X'08')
1	Compound type identifier; 1 = Factored; 2 self describing fields where the first is the factor used, and the second is the resultant value
1	Length of the first field, including this byte
1	Data type identifier
n	First data field as identified by the preceding data type field; for example Binary(8)
1	Length of the next field, including this byte
1	Data type identifier
n	Next data field as identified by the preceding data type field; for example Binary(64)

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 be used to identify subgroups within a group. The Begin and End Resource groups are never followed by data. Table 13 shows SFIs that identify Begin and End resource groups.

Table 13. 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'027500' - X'027580'	MEDINF - within CONTROL
X'028000' - X'028080'	MESSAGE
X'029000' - X'029080'	MNTMSG - within CONTROL

Table 13. Begin and End Group Structured Field Introducers (continued)

Begin - End IDs	Resource Group
X'02A000' - X'02A080'	MOVES - within CONTROL
X'03A000' - X'03A080'	OPENRULE within CONTROL
X'02B000' - X'02B080'	OPTION - within CONTROL
X'02C000' - X'02C080'	OWNER
X'02D000' - X'02D080'	PRODUCT
X'039000' - X'039080'	PRODVOL - within PRODUCT
X'03B000' - X'03B080'	PRITITION within CONTROL
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 14 provide return codes 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 that are returned from OAM. The RSNC and RTNC SFIs are used for return and reason codes that are from another system service.

When the DFSMSrmm API builds a SYSRETC group for an error reported by a system service, look for additional information that is available from system messages in places like the operator terminal, SYSTSPRT, job log, and SYSLOG data set.

Subcommands are described using standard DFSMSrmm abbreviations. For example, AV is for ADDVOLUME as shown in Table 3 on page 2. The SFI values are enclosed in single quotes (') to signify that they are 8-byte hexadecimal values. Two spaces are included in the IDs for readability.

Table 14. Reason and Return Code SFIs

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'400000'	FRC	12	Binary(32)	Function return code	AV CV DV GV
X'401000'	FRS	12	Binary(32)	Function reason code	AV CV DV GV
X'402000'	RSNC	12	Binary(32)	Reason code	Any subcommand
X'403000'	RTNC	12	Binary(32)	Return code	Any subcommand

Table 14. Reason and Return Code SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'404000'	SVCN	16	Character (variable length)	Service name	Any subcommand

SFIs for Messages and Message Variables

The SFIs described in Table 15 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 60.

Table 15. 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(32)	Number of entries Min 0, Max 10-digit	As previously defined
X'054000'	KEYF	65	Character (variable length)	Key from	SD SV
X'054200'	KEYT	65	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
X'057000'	CONT	92	Character (variable length)	SEARCH Continue information	All search subcommands

SFIs for Subcommand Output Data

The SFIs described in Table 16 on page 87 introduce subcommand output data in your output buffer. These SFIs are always grouped within a pair of Begin and End Resource group SFIs.

This 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 16. Command SFIs

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'800500'	ABND	9	Binary(8)	Closed by Abend 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'801800'	ACS	9	Binary(8)	SMSACS 0=NO 1=YES	LC
X'802000'	ACT	9	Bit(8)	Actions on release '80'=SCRATCH '40'=REPLACE '20'=INIT '10'=ERASE '08'=RETURN '04'=NOTIFY For LC VLPOOL X'00', X'04'	LC LV SV(e)
X'803001'	ADL	48	Character (variable length)	Address line. The SFI is incremented by one for each ADL line that is found. (X'803001' - X'803003')	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(15)	SMF audit record type: 128-255, 42, or 0	LC
X'808000'	AVL	9	Bit(8)	Volume availability '40'=PENDING_RELEASE '20'=VITAL_RECORD '08'=ON_LOAN '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 alphanumeric bin number	LV SV(e)
X'80B000'	BKPP	16	Character (Variable length)	Backup procedure name	LC

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'80C000'	BLKC	12	Binary(32)	Block count	LD SD(e)
X'80D000'	BLKS	12	Binary(32)	Block size	LD SD(e)
X'80D030'	BLKT	12	Binary(32)	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'812900'	CDSQ	9	Binary(8)	Control data set ENQ 0=Disabled 1=Enabled	LC
X'812A00'	CDSU	10	Binary(15)	Control data set percentage used	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'816900'	CMDD	9	Binary(8)	Command Authorization - based on DSN: 0=No 1=Yes	LC
X'8169A0'	CMDO	9	Binary(8)	Command Authorization - based on owner: 0=No 1=Yes	LC
X'817000'	CNT	12	Binary(32)	Bin, rack, or volume count: Min 0, Max 5-digit	AB AR AV DB DR
X'817820'	CPGM	16	Character (fixed length)	Creating program name	LD SD(e)
X'818000'	CRP	12	Binary(32)	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

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'819200'	CSHN	71	Character (variable length)	Client/server host name 1-to-63 alphanumeric characters including hyphen, period, and blank	LC
X'819250'	CSIP	23	Character (variable length)	Client IP address 1-to-15 numeric characters including period and blank	LC
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'81A600'	DBIN	14	Character (fixed length)	Numeric: 0-999999 or 6 alphanumeric character destination bin number	LV
X'81A700'	DBMN	16	Character (variable length)	Destination bin media name	LV
X'81B000'	DBN	12	Binary(32)	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'	DLR/ DLRJ	12	Packed decimal Julian date format	Date last referenced/read	LD LV SD(e) LS SS(e) SV(e)
X'823700'	DLTD	9	Binary(8)	Deleted by disposition processing: 0=NO 1=YES	LD SD(e)

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'824000'	DLWJ	12	Packed decimal Julian date format	Date last written	LD LV SD(e) SV(e)
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(32)	Default retention period: Min 0, Max 4-digit	LC
X'828000'	DSC	12	Binary(32)	Data set count: Min 0, Max 4-digit	LV SV
X'829000'	DSEQ	12	Binary(32)	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=NO 1=YES	LV SV
X'82B030'	DSS6	22	Compound (Binary(8) Factor, Binary(64) Value)	Data set size, Factor: 0=bytes 1=KB 2=MB 3=GB 4=TB Value: Minimum value = 0.	LD SD(e)
X'82B200'	DSTT	9	Binary(8)	Destination type 0=SHELF 1=STORE_BUILTIN 2=MANUAL 3=AUTO 4=STORE_BINS 5=STORE_NOBINS	LV
X'82BB00'	DSYS	16	Character (variable length)	Creating system ID	LV, SV(e)
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'82D500'	EBIN	9	Binary(8)	Extended bin enable status 0=DISABLED 1=ENABLED	LC
X'82DFF0'	EML	71	Character (variable length)	Owner's e-mail address, 1 to 63 characters	LO SO
X'82E000'	EMN	16	Character (variable length)	Owner's node	LO

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
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(32)	Free bins in DISTANT location Min 0, Max 6-digit	LC
X'833000'	FILE	12	Binary(32)	Physical file sequence Min 1, Max 4-digit	LD SD
X'834000'	FLB	12	Binary(32)	Free bin numbers in LOCAL location: Min 0, Max 6-digit	LC
X'835000'	FOR	28	Character (variable length)	Owner's forename	LO
X'836000'	FRB	12	Binary(32)	Free bin numbers in REMOTE location: Min 0, Max 6-digit	LC
X'837000'	FRK	12	Binary(32)	Free rack numbers in library: Min 0, Max 10-digit	LC
X'837800'	GDGC	9	Binary(8)	GDG CYCLEBY: 0=Generation 1=Create order	LC
X'837805'	GDGD	9	Binary(8)	GDG DUPLICATE: 0=Bump from sub chain 1=Drop from chain 2=Keep 3=Count	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'839200'	HLOT	9	Binary(8)	Home location type 0=SHELF 1=STORE_BUILTIN 2=MANUAL 3=AUTO 4=STORE_BINS 5=STORE_NOBINS	LV
X'83A000'	INTR	9	Binary(8)	Volume intransit status: 0=NO 1=YES	LV SV
X'83B000'	IPL	9	Binary(8)	Date check required on IPL: 0=NO 1=YES	LC
X'83B830'	IRRM	9	Binary(8)	Managed by IRMM), 0=NO 1=YES	LV, SV (e)
X'83C000'	ITL	16	Character (variable length)	Telephone number	LO
X'83CA00'	JBDT	12	Packed decimal Julian date	Last Journal Backup Date	LC
X'83CB00'	JBTM	12	Packed decimal time format	Last Journal Backup Time	LC

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'83D000'	JDS	52	Character (variable length)	Journal name	LC
X'83E000'	JRNF	10	Binary(15)	JOURNALFULL parmlib value: 0 - 99	LC
X'83EA00'	JRNS	9	Binary(8)	Journal status: 0=Disabled 1=Enabled 2=Locked	LC
X'83ED00'	JRNT	9	Binary(8)	Journal transaction: 0=No 1=Yes	LC
X'83F000'	JRNU	10	Binary(15)	Journal percentage used: 0 - 100	LC
X'83F500'	KEL1	72	Character (variable length)	Key encryption key label 1	LV, SV(e)
X'83F505'	KEL2	72	Character (variable length)	Key encryption key label 2	LV, SV(e)
X'83F520'	KEM1	13	Character (variable length)	Key encoding mechanism for key label 1: LABEL or HASH	LV, SV(e)
X'83F525'	KEM2	13	Character (variable length)	Key encoding mechanism for key label 2: LABEL or HASH	LV, SV(e)
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(32)	Bin numbers in LOCAL location Min 0, Max 6-digit	LC
X'842000'	LCID	16	Last change user ID	Character (variable length)	LV SV(e)
X'843000'	LCT	10	Binary(15)	Default lines per page Min 10, Max 3-digit	LC
X'843100'	LCTK	12	Binary (31)	Local tasks binary value	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	12	Character (fixed length)	Last drive	LD SD(e) LV SV(e)
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

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'849000'	LDMT	9	Binary(8)	Location management type: 0=UNDEFINED 1=BIN 2=NOBINS	LC
X'84A000'	LDPR	12	Binary(32)	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 (variable length)	Location	LB LR LS LV SB SR SS SV
X'84E000'	LOCT	9	Binary(8)	Location type 0=SHELF 1=STORE_BUILTIN 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(32)	Logical record length: Min 0, Max 5-digit	LD SD(e)
X'850000'	LRK	12	Binary(32)	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 3=Label version 3 4=Label version 4	LV SV
X'850A00'	LVN	9	Binary(8)	Required label version: 0=No version specified 3=Label version 3 4=Label version 4	LV SV
X'851000'	MC	16	Character (variable length)	Management class	LD SD(e)
X'851400'	MDNF	16	Character (8)	Media Information Name	LV SV(e) LC
X'851980'	MDRA	12	Binary(32)	MEDINF replace policy for age	LC
X'8519C0'	MDRP	12	Binary(32)	MEDINF replace policy for permanent errors	LC
X'8519E0'	MDRT	12	Binary(32)	MEDINF replace policy for temporary errors	LC
X'8519F0'	MDRW	12	Binary(32)	MEDINF replace policy for write mount count	LC
X'851A00'	MDRX	16	Character (8)	External Recording Technology	LV SV(e) LC

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
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'853400'	MDTX	16	Character (8)	External Media Type	LV SV(e) LC
X'854000'	MEDA	9	Binary(8)	Media special attributes: 0=NONE 1=RDCOMPAT	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 5=384TRK 6=EFMT1 7=EFMT2 8=EEFMT2 9=EFMT3 10=EEFMT3	LV SV LC
X'858000'	MEDT	9	Binary(8)	Media type: 0=UNDEFINED 1=CST 2=ECCST 3=HPCT 4=EHPCT 5=ETC/MEDIA5 6=EWTC/MEDIA6 7=EETC/MEDIA7 8=EEWTC/MEDIA8 9=EXTC/MEDIA9 10=EXWTC/MEDIA10	LV SV LC
X'859000'	MFR	16	Character (variable length)	Source location name	LC
X'85A000'	MID	20	Character (variable length)	Mount message ID	LC
X'85A500'	MIV	14	Character (fixed length)	Moving-in volume	LB SB
X'85A900'	MOV	14	Character (fixed length)	Moving-out volume	LB SB
X'85B000'	MOV M	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(32)	Maximum retention period: Min 0, Max 4-digit -1 (negative) means unlimited retention.	LC

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'85E000'	MSGF	9	Binary(8)	Message text case: 0=MIXED 1=UPPER	LC
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=NOTRIS 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'865200'	NLOT	9	Binary(8)	Required location type 0=SHELF 1=STORE_BUILTIN 2=MANUAL 3=AUTO 4=STORE_BINS 5=STORE_NOBINS	LV
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)

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'86C200'	OLOT	9	Binary(8)	Old location type 0=SHELF 1=STORE_BUILTIN 2=MANUAL 3=AUTO 4=STORE_BINS 5=STORE_NOBINS 6=IN_CONTAINER	LV
X'86D000'	OPL	10	Binary(15)	Position of rack number or pool ID Min 1, Max 3-digit	LC Position in the message.
X'86E000'	OPM	9	Binary(8)	Operating mode 1=M 2=R 3=W 4=P	LC
X'86E8A0'	ORIA	9	Binary(8)	Input action: 0=ACCEPT 1=IGNORE 2=REJECT	LC
X'86E8A8'	ORII	9	Bit(8)	Input IGNORE condition (BY): X'80'=SPECIFIC X'40'=NONSPECIFIC X'C0'=ANY	LC
X'86E8B8'	ORIR	9	Bit(8)	Input REJECT condition (BY): X'80'=SYSID X'40'=CATLG	LC
X'86EA00'	OROA	9	Binary(8)	Output action: 0=ACCEPT 1=IGNORE 2=REJECT	LC
X'86EA08'	OROI	9	Bit(8)	Output IGNORE condition (BY): X'80'=SPECIFIC X'40'=NONSPECIFIC X'C0'=ANY	LC
X'86EA18'	OROR	9	Bit(8)	Output REJECT condition (BY): X'80'=SYSID X'40'=CATLG	LC
X'86EF08'	ORTP	9	Binary(8)	Type of open rule entry: 0=RMM 1=NORMM	LC
X'86EF80'	ORVS	14	Character (variable length)	Volume range start	LC
X'86EF85'	ORVL	14	Character (variable length)	Volume serial number, specific or generic	LC
X'86EF8F'	ORVE	14	Character (variable length)	Volume range end	LC
X'86F000'	OVL	10	Binary(15)	Position of volume serial number: Min 1, Max 3-digit	LC Position in the message.
X'86F500'	OVOL	14	Character (fixed length)	Old volume	LB SB
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	LD LV SD SV(e)

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'871800'	PACS	9	Binary(8)	PREACS 0=NO 1=YES	LC
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 LR SR
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(32)	Permanent read errors: Min 0, Max 5-digit	LV SV(e)
X'87A000'	PRF	9	Binary(8)	Pool definition RACF® (A component of the Security Server for z/OS) option: 0=NO 1=YES	LC
X'87B000'	PRTY	12	Binary(32)	Priority: Min 0, Max 4-digit	LS SS
X'87C000'	PSFX	10	Character (fixed length)	Parmlib member suffix	LC
X'87C010'	PSF2	10	Character (fixed length)	Second parmlib member suffix	LC
X'87D000'	PSN	16	Character (variable length)	Pool definition system ID	LC
X'87DB00'	PTNA	9	Binary(8)	NOSMT action for partition entry: 0=ACCEPT 1=IGNORE	LC
X'87DB0C'	PTNL	16	Character (variable length)	Location name	LC
X'87E000'	PTP	9	Binary(8)	Pool definition pool type: 0=SCRATCH 1=RACK	LC

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'87EB80'	PTSA	9	Binary(8)	SMT action for partition entry: 0=ACCEPT 1=IGNORE	LC
X'87EBA8'	PTTP	9	Binary(8)	Type of partition entry: 0=RMM 1=NORMM	LC
X'87EC00'	PTVS	14	Character (variable length)	Volume range start	LC
X'87EC08'	PTVL	14	Character (variable length)	Volume serial number, specific or generic	LC
X'87EC0F'	PTVE	14	Character (variable length)	Volume range end	LC
X'87F000'	PVL	14	Character (fixed length)	Previous volume: 1 - 6 character	LV SV(e)
X'880000'	PWT	12	Binary(32)	Permanent write errors: Min 0, Max 5-digit	LV SV(e)
X'881000'	RBN	12	Binary(32)	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 SV(e)
X'882000'	RCF	9	Binary(8)	Installation RACF support: 1=N 2=P 3=A 4=C	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
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

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'88E000'	RTM	12	Packed decimal time format	Last control data set extract time	LC
X'88E500'	RUB	9	Binary(8)	Reuse bin at 0=CONFIRMMOVE 1=STARTMOVE	LC
X'890000'	SC	16	Character (variable length)	Storage class name	LD SD(e)
X'891000'	SCRM	9	Binary(8)	Binary value 0=Auto 1=manual	LC
X'892000'	SCST	9	Bit(8)	Security class status '80'=SMF '40'=MSGOPT '20'=ERASE	LC
X'894000'	SC1	12	Binary(32)	Storenummer Min 1, Max 5-digit	LS SS 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(32)	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
X'8A1000'	SOST	12	Packed decimal time format	Last expiration processing start time	LC
X'8A1A00'	SRHN	71	Character (variable length)	Server host name 1-to-63 alphanumeric characters including hyphen, period, and blank	LC
X'8A1A30'	SRIP	23	Character (variable length)	Server IP address 1-to-15 numeric characters including period and blank	LC
X'8A1A50'	SRPN	12	Binary (31)	Server number binary value	LC

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'8A1AF0'	SRTK	12	Binary (31)	Server tasks binary value	LC
X'8A2000'	SSM	10	Binary(15)	SMF security record type: 128-255, 42, or 0	LC
X'8A2500'	SSTY	9	Binary (8)	Subsystem type 0=Standard system 1=Client system 2=Server system	LC
X'8A3000'	STEP	16	Character (variable length)	Step name	LD SD(e)
X'8A3800'	STVC	12	Binary(32)	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'8A6800'	TLR	12	Packed decimal time format	hhmmsstC, where hhmmsst is the time in hours, minutes, seconds, and tenths of seconds and C is a standard packed-decimal sign character.	LS SS(e)
X'8A7000'	TRD	12	Binary(32)	Temporary read errors Min 0, Max 5-digit	LV SV(e)
X'8A7900'	TVXP	9	Binary(8)	Tape volume exit purge option: 0=RELEASE 1=EXPIRE 2=NONE	LC
X'8A8000'	TWT	12	Binary(32)	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'8A9E00'	TZ	12	Binary(32)	Signed number; the offset from common time in seconds. When non-zero, use this value to adjust all dates and times from the DFSMSrmm systems' local time to common time.	All
X'8AA000'	UDTJ	12	Packed decimal Julian date format	Late update date	LC
X'8AB001'	UID	16	Character (variable length)	User ID. The SFI is incremented by one for each UID that is found. (X'8AB001'-X'8AB00C')	LV SV(e)
X'8AC000'	UNC	9	Binary(8)	Uncatalog option: 0=N 1=Y 2=S	LC

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'8AD000'	USEC	12	Binary(32)	Volume use count: Min 0, Max 5-digit	LV SV(e)
X'8AE000'	USEM	12	Binary(32) unsigned	Volume usage (KB): Min 0, Max 4294967295. 4294967295 indicates that USE6 must be used.	LV SV(e)
X'8AE030'	USE6	22	Compound (Binary(8) Factor, Binary(64) Value)	Volume usage, Factor: 0=bytes 1=KB 2=MB 3=GB 4=TB Value: Minimum value = 0.	LV SV(e)
X'8AE600'	UTC	9	Binary(8)	Common Time: 0=DISABLED 1=ENABLED	LC
X'8AE800'	UTM	12	Packed decimal time format	Late update time	LC
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 3=OFF	LC
X'8B0800'	VANX	9'	Binary(8)	Next VRS type: 0=Undefined 1=Next 2=And	LS SS
X'8B0B00'	VCAP	12	Binary(32)	Volume/Media capacity	LV SV(e) LC
X'8B1000'	VCHG	9	Binary(8)	VRSCCHANGE value: 0=INFO 1=VERIFY	LC
X'8B2000'	VDD	10	Binary(15)	VRS delay days: Min 0, Max 99	LS SS(e)
X'8B2800'	VDRA	9	Binary(8)	VRSDROP action: 0=FAIL 1=INFO 2=WARN 3=OFF	LC
X'8B2802'	VDRC	12	Binary(32)	VRSDROP count	LC
X'8B280F'	VDRP	10	Binary(15)	VRSDROP percent	LC
X'8B3000'	VD TJ	12	Packed decimal time format	Last inventory management processing date	LC
X'8B4000'	VER	14	Character (variable length)	Software produce version, release, modification vvrmm	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(32)	Number of volumes: Min 0, Max 3-digit	LO LP SP
X'8B7000'	VM	9	Binary(8)	VM use: 0=NO 1=YES	LV SV(e)

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
X'8B8000'	VMIN	12	Binary(32)	VRSMIN count value: Min 0, Max 6-digit	LC
X'8B9000'	VMV	16	Character (variable length)	VRS management value	LD SD(e)
X'8B9100'	VWMC	12	Binary(32)	Volume write mount count	LV, SV(e)
X'8B9E00'	VNDR	16	Character (8)	Vendor information	LV, SV(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	AV CV GV LB LD LP LR LV SB SD SP SR SV
X'8BC200'	VOLT	9	Binary(8)	Volume type: 0=PHYSICAL 1=LOGICAL 2=STACKED	LV SV(e)
X'8BCD00'	VOL1	14	Character (fixed length)	VOL1 label volume serial number	LV SV(e)
X'8BC300'	VPCT	9	Binary(8)	Volume percent full	LV SV(e)
X'8BD000'	VRC	12	Binary(32)	Vital record count: Min 1, Max 5-digit	LS SS SS(e)
X'8BD500'	VREA	9	Binary(8)	VRSRETAIN action: 0=FAIL 1=INFO 2=WARN 3=OFF	LC
X'8BD502'	VREC	12	Binary(32)	VRSRETAIN count	LC
X'8BD50F'	VREP	10	Binary(15)	VRSRETAIN percent	LC
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: 1=NEW	LC
X'8C0000'	VRSR	9	Binary(8)	VRS retained status: 0=NO 1=YES	LD SD 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

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
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'8C4300'	WORM	9	Binary(8)	Volume is WORM: 0=NO 1=YES	LV, SV (e)
X'8C4500'	WWID	32	Character (24)	World-wide identifier	LV, SV (e)
X'8C5000'	XDC	9	Binary(8)	Expiration date check: 0=NO 1=YES 2=OPERATOR	LC
X'8C5D00'	XDRA	9	Binary(8)	EXPDTDROP action: 0=FAIL 1=INFO 2=WARN 3=OFF	LC
X'8C5D02'	XDRC	12	Binary(32)	EXPDTDROP count	LC
X'8C5D0F'	XDRP	10	Binary(15)	EXPDTDROP percent	LC
X'8C6000'	XDTJ	12	Packed decimal Julian date format	Expiration date	LC LD LV SD SV
X'8C7000'	XTM	12	Packed decimal time format	Last inventory management expiration time	LC
X'8C7800'	X100	9	Binary(8)	EDG_EXIT100 installation exit status: 0 Exit is not defined or no exit modules exist 1 At least one active exit module exists 2 One or more exit modules exist, but none is active	LC
X'8C7801'	X200	9	Binary(8)	EDG_EXIT200 installation exit status: 0 Exit is not defined or no exit modules exist 1 At least one active exit module exists 2 One or more exit modules exist, but none is active	LC
X'8C7802'	X300	9	Binary (8)	EDG_EXIT300 installation exit status: 0 Exit is not defined or no exit modules exist 1 At least one active exit module exists 2 One or more exit modules exist, but none is active	LC
X'8C8000'	2JBN	16	Character (variable length)	Secondary VRS jobname mask	LD SD(e)
X'8C9000'	2NME	16	Character (variable length)	Secondary VRS mask	LD SD(e)

Table 16. Command SFIs (continued)

SFI Number	SFI Name	SFI Length	SFI Data Type	Data Description	Subcommand
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 17 lists the structured field introducers by DFSMSrmm TSO subcommand.

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 17. 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
ADDDOWNER	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
CHANGEVRS	ENTN MSGL MSGN 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 MIV MOV MEDN MSGL MSGN OVOL RCK RSNC RST RTNC SVCN TZ VOL
LISTCONTROL ACTIONS	ACT AST RC
LISTCONTROL CNTL	ACS AUD BDT BTM CDSQ CDSU CSHN CSIP CSVE DBN CDS CSDT CSTM DDT DRP DTE DTM EBIN FBP FCSP FDB FEP FKP FLB FRB FRK FRP FSP FTP FVP FXP IPL JBDT JBTM JDS JRNS JRNU LBN LCT LRK MDS MDT MRP MTM MTP NOT OPM PACS RBN RC RCF RDT RTM RUB SAT SDT SID SLM SOSD SOSP SOST SSM STM UDT UTC UTM VDT VTM XDTJ XTM X100 X200 X300

Table 17. Structured Field Introducers by Subcommand (continued)

Subcommand	Structured Field Introducers
LISTCONTROL LOCDEF	LDDF LDLC LDLT LDMN LDMT LDPR RC
LISTCONTROL MNTMSG	MID OPL OVL RC SMI
LISTCONTROL MEDINF	MDNF MDRA MDRP MDRT MDRW MDRX MDTX MEDR MEDT VCAP
LISTCONTROL MOVES	MFR MST MTO MTY
LISTCONTROL OPENRULE	ORIA ORII ORIR OROA OROI OROR ORTP ORVE ORVL ORVS
LISTCONTROL OPTION	ACCT AUD BLP BKPP CATS CDS CMDD CMDO CRP DRP DSPD DSPM DTE GDGC GDGD IPL JDS JRNF JRNT LCT LCTK MDS MEDN MOP MRP MSGF MVBY OPM NOT PDAC PDA PDAC PDAL PDAS PSFX PSF2 RC RCF RTBY RUB SID SLM SMP SMUC SMUE SMUS SOSP SRHN SRIP SRPN SRTK SSM SSTY TVXP UNC VACT VCHG VDRA VDRC VDRP VMIN VREA VREC VREP VRJ VRSL XDRA XDRC XDRP
LISTCONTROL PRITION	PTNA PTNL PTSA PTPP PTVE PTVL PTVS
LISTCONTROL REJECT	GRK RC TAC
LISTCONTROL SECCLS	CLS ERS MSG NME RC SEC SMF
LISTCONTROL SECLEVEL	CLS DNM ERS MSG NME RC SEC SMF
LISTCONTROL VLPOOL	ACT MEDN MOP PDS PID PLN PRF PSN PTP SCRML XDC
LISTDATASET	ABND BLKC BLKS BLKT CDTJ CJBN CLS CPGM CTLG CTM DC DD DEV DLRJ DLTD DLWJ DPCT DSEQ DSN DSS6 ENTN FILE LDD LDEV LINE LPGM LRCL LSTP MC MSGL MSGN NME OWN OXDJ RCFM RSNC RTDJ RTNC SC SG STEP SVCN SYS TZ VJBN VNME VOL VRSR VSCD VSCN VTYP XDTJ 2JBN 2NME 2SCD 2SCN
LISTOWNER	ADL DPT EML EMN EMU ENTN ETL FOR ITL LINE MSGL MSGN OWN RSNC RTNC SUR SVCN TZ VLN
LISTPRODUCT	ENTN FCD LINE MSGL MSGN OWN PDSC PNME PNUM RCK RSNC RTNC SVCN TZ VER VLN VOL
LISTTRACK	ENTN LINE LOC MEDN MSGL MSGN PID RCK RSNC RST RTNC SVCN VOL
LISTVOLUME	ACN ACT ADTJ ATM AVL BIN BMN CDTJ CJBN CLS CTM CTNR DBIN DBMN DEN DESC DEST DLRJ DLWJ DSC DSN DSR DSTT ENTN FCD HLOC HLOT INTR KEL1 KEL2 KEM1 KEM2 LBL LCID LDEV LINE LOAN LOC LOCT LVC LVN MDNF MDRX MDTX MEDA MEDC MEDN MEDR MEDT MOVN MSGL MSGN MVS NLOC NLOT NME NVL OAC OBMN OBN OCE OLOC OLOT OWN OXDJ PEND PNUM PRD PVL PWT RBYS RCK RSNC RTDJ RTNC SDTJ SEQ SG STVC SVCN TRD TWT TZ UID01...UID12 USEC USEM USE6 VAC VCAP VER VM VMIN VNDR VOL VOLT VOL1 VPCT VRSI VRXI VST VWMC WORM WWID XDTJ
LISTVRS	DDTJ DESC DLRJ ENTN LINE LOC MSGL MSGN NVRS OWN PRTY RET RSNC RTNC SC1 SVCN TLR TYP TZ VANX VDD VJBN VRC VRS VRSI VRXI
SEARCHBIN	CONT ENTN LINE LOC MEDN MIV MOV MSGL MSGN OVOL RCK RSNC RST RTNC SVCN TZ VOL
SEARCHDATASET	CDTJ CONT CTM DSN ENTN FILE KEYF KEYT LINE MSGL MSGN OWN OXDJ RSNC RTNC SVCN VOL XDTJ
SEARCHDATASET(EXPAND=YES)	The same SFIs as the LISTDATASET subcommand.

Table 17. Structured Field Introducers by Subcommand (continued)

Subcommand	Structured Field Introducers
SEARCHOWNER	ADL CONT DPT EML EMN EMU ETL FOR ITL OWN SUR TZ VLN
SEARCHPRODUCT	CONT 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	CONT ENTN LINE LOC MEDN MSGL MSGN PID RCK RSNC RST RTNC SVCN VOL
SEARCHVOLUME	ADTJ AVL CONT DESC DSC DSR ENTN HLOC INTR KEYF KEYT LBL LINE LOAN LOC LVC LVN MDNF MDRX MDTX MEDA MEDC MEDN MEDR MEDT MSGL MSGN OWN PEND RCK RSNC RTDJ RTNC SEQ SVCN TYPF TYPT VCAP VOL VST XDTJ
SEARCHVOLUME(EXPAND=YES)	The same SFIs as the LISTVOLUME subcommand.
SEARCHVRS	CONT DDTJ ENTN LINE LOC MSGL MSGN NVRS OWN PRTY RET RSNC RTNC SVCN VANX VJBN 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 Figure 53 “EDGXCI: Parameter List”
- The structured field definitions generated by the EDGXSF macro as shown in “EDGXSF: Structured Field Definitions” on page 110

EDGXCI: Parameter List

The mapping of the parameter list is generated by the list form of the EDGXCI macro.

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_XMULTI DS XL1            ++ XMULTI
| MYPL_XMULTI_NO EQU 0          ++ XMULTI.NO KEYWORD
| MYPL_XMULTI_YES EQU 1         ++ XMULTI.YES KEYWORD
| MYPL_XRSV0001 DS CL7          ++ RESERVED XRSV0001
| MYPL_XRSV0002 DS CL4          ++ RESERVED XRSV0002
| MYPL_XRSV0003 DS CL8          ++ RESERVED XRSV0003
| MYPLL    EQU    *-MYPL        ++ LENGTH OF PLIST
```

Figure 53. 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” on page 111
- “EDGXSF Labeling Conventions” on page 112

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. Table 18 and Table 19 on page 112 show 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 83.

Table 18. Output Buffer and Structured Field Introducers

Offsets					
Dec	Hex	Type	Len	Name(Dim)	Description
0	(0)	STRUCTURE	*	XSF_OUTBUF	Output Buffer
0	(0)	SIGNED	4	XSF_OUTBUF_BUFLNG	Output Buffer Length
4	(4)	SIGNED	4	XSF_OUTBUF_RQDLNG	Required Buffer Length
8	(8)	SIGNED	4	XSF_OUTBUF_DATA LNG	Length of Output Data
12	(C)	CHARACTER	*	XSF_OUTBUF_FIELDS	Start of Structured Fields
Structured Field Introducers for Structured Fields					
0	(0)	STRUCTURE	*	XSF_SFI	Structured Field Introducers
0	(0)	CHARACTER	8	XSF_SFI_HD	
0	(0)	SIGNED	2	XSF_SFI_LENGTH	Length
2	(2)	CHARACTER	3	XSF_SFI_ID	Identifier
2	(2)	CHARACTER	2	XSF_SFI_IDVAL	Identifier value
4	(4)	CHARACTER	1	XSF_SFI_IDQUAL	Identifier qualifier
5	(5)	UNSIGNED	1	XSF_SFI_TYPE	Type
6	(6)	CHARACTER	1	*	Reserved
7	(7)	UNSIGNED	1	XSF_SFI_DTYPE	Data type
8	(8)	CHARACTER	*	XSF_SFI_DATA	Start of data
Compound SFI definition					
8	(8)	STRUCTURE	14	XSF_SFI_COMPTYPE1	Compound section
8	(8)	CHARACTER	6	XSF_SFI_COMPDATA	
8	(8)	CHARACTER	6	XSF_SFI_COMPHDR	Compound Hdr
8	(8)	CHARACTER	6	XSF_SFI_COMPENT	Compound Entry
8	(8)	UNSIGNED	1	XSF_SFI_COMPTYPE	Compound type
9	(9)	CHARACTER	3	XSF_SFI_FIELD1	
9	(9)	UNSIGNED	1	XSF_SFI_LEN1	Length of first field
10	(A)	UNSIGNED	1	XSF_SFI_DTYP1	Type of first field
11	(B)	UNSIGNED	1	XSF_SFI_FACTOR	Factor for second field
12	(C)	CHARACTER	2	XSF_SFI_FIELD2	
12	(C)	UNSIGNED	1	XSF_SFI_LEN2	Length of second field
13	(D)	UNSIGNED	1	XSF_SFI_DTYP2	Type of second field
14	(E)	CHARACTER	8	XSF_SFI_COMPVAL	The Value

Table 19. Constants for EDGSFI

Len	Type	Value	Name	Description
Data Types (XSF_SFI_DTYPE, XSF_SFI_DTYP1, XSF_SFI_DTYP2)				
1	HEX	00	XSF_SFI_DTYPE_UNDEF	Undefined data
1	HEX	01	XSF_SFI_DTYPE_CHAR_FIX	n-byte character
1	HEX	02	XSF_SFI_DTYPE_BITFLAG	1-byte bit flag byte (8 bits)
1	HEX	03	XSF_SFI_DTYPE_BIN8	1-byte (hex) value
1	HEX	04	XSF_SFI_DTYPE_BIN15	2-byte hex value
1	HEX	05	XSF_SFI_DTYPE_BIN31	4-byte hex value
1	HEX	06	XSF_SFI_DTYPE_BIN64	8-byte hex value
1	HEX	07	XSF_SFI_DTYPE_CHAR_VAR	Variable length character
1	HEX	08	XSF_SFI_DTYPE_COMPOUND	Compound SFI
1	HEX	09	XSF_SFI_DTYPE_JDATE	4-byte packed decimal date <i>yyyymmddc</i> - c is sign
1	HEX	0A	XSF_SFI_DTYPE_TIME	4-byte packed decimal time <i>hhmmssct</i> - c is sign
Compound Types (XSF_SFI_CompType)				
1	HEX	00	XSF_SFI_COMPTYPE_UNDEF	Undefined type
1	HEX	01	XSF_SFI_COMPTYPE_FACTOR	Factored type
Factors (XSF_SFI_Factor)				
1	HEX	00	XSF_SFI_FACTOR_BYTES	Value is in Bytes
1	HEX	01	XSF_SFI_FACTOR_KB	Value is in KBytes
1	HEX	02	XSF_SFI_FACTOR_MB	Value is in MBytes
1	HEX	03	XSF_SFI_FACTOR_GB	Value is in GBytes
1	HEX	04	XSF_SFI_FACTOR_TB	Value is in TBytes

EDGXSF Labeling Conventions

This topic 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 this format:

- XSF_SFI_ID_xxxx and XSF_xxxx_LENGTH
- XSF_SFI_ID_Exxxx and XSF_Exxxx_LENGTH

Figure 54 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 54. Mapping of the Begin and End ACCESS Group

The VOL and VRS groups are defined using this format:

- XSF_SFI_ID_XXX and XSF_XXXGRP_LENGTH
- XSF_SFI_ID_EXXX and XSF_EXXXGRP_LENGTH

Figure 55 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 55. Mapping of the Begin and End VOL Group

Labeling: SFIs that Introduce Data

SFIs introduce data and are defined using this format:

- XSF_SFI_XXXX_ID
- XSF_XXXX_LENGTH
- XSF_XXXX_DTYPE

Figure 56 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 56. Mapping of the ATM SFI

Labeling: Flags

Output data for some SFIs are defined as bit flags using this format:

XSF_XXXX_FLAG_name.

Figure 57 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 57. Mapping of the ACT SFI

Labeling: Bin(8) Data

Output data for some SFIs are defined as one-byte binary numbers using this format: XSF_XXXX_DATA_name.

Figure 58 on page 114 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'
XSF_LOCT_DATA_INCTNR EQU X'06'
* *****

```

Figure 58. Mapping of the LOCT SFI

Unlabeled Data

These 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 topic 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 59 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 55. This format is used:

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

```
0000 0000100000000000 0000008400080260 00000000001A82A0 00000007D9D4D4E4
0020 E2C5D94BC6C9C5D3 C44BE3C5E2E3000E 8BC000000001E5D6 D3F0F0F1000F8700
0040 00000007D9D4D4E4 E2C5D9000C8A9E00 000005FFFF9D9000 0C81300000000920
0060 05320F000C81A000 00000A0658226F00 0C83300000000500 0000010008026080
0080 0000000000000000 0000000000000000 0000000000000000 0000000000000000

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

Figure 59. 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 59 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.

```
00000084
```

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 59 on page 115 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 59 on page 115 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 59 on page 115 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 these guidelines:

1. Base the XSF_OUTBUF definition in macro EDGXSF as shown in Figure 60 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_DATA LNG	DS	1FL4	Length of Output Data
XSF_OUTBUF_FIELDS	DS	0C	Start of Structured Fields

Figure 60. Output Buffer Definition

2. Base the XSF_SFI definition in macro EDGXSF as shown in Figure 61 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

Note: XSF_SFI_DATA can contain compound data with an internal structure of:

```
XSF_SFI_CompType
XSF_SFI_LEN1
XSF_SFI_DTYPE1
XSF_SFI_Factor
XSF_SFI_LEN2
XSF_SFI_DTYPE2
XSF_SFI_Value
```

Figure 61. SFI Definition

3. Find the type of structured field you are processing by using the two-byte structured field identifier at XSF_SFI_IDVAL. 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 83 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_DATA LNG.
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.

| **Note:** This is true only if you did not specify MULTI=YES in your call to the
| API. If you use MULTI=YES, your output buffer may contain more
| than one resource group.

- 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 60 on page 117 and Figure 61 on page 117:

- 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.

Appendix E. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

<http://www.ibm.com/systems/z/os/zos/bkserv/>

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
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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

The licensed program described in this 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.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Trademarks

DFSMSrmm
IBM
IBMLink
RACF
z/OS
z/VM

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Apache Tomcat and Tomcat are trademarks of the Apache Software Foundation in the United States, other countries, or both.

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

Index

Numerics

2JBN ('8C8000') - Secondary VRS
Jobname Mask 103
2NME ('8C9000') - Secondary VRS
Mask 103
2SCD ('8CA000') - Secondary VRS
Subchain Start Date 104
2SCN ('8CB000') - Secondary VRS
Subchain Name 104

A

abbreviations for subcommands 2
ABND ('800800') - closed by Abend 87
accessibility 119
account number SFI 87
accounting source SFI 87
ACCT ('800800') - Accounting Source 87
ACN ('801000') - Account Number 87
ACS ('801800') - SMSACS 87
ACT ('802000') - Actions on release 87
Action Status SFI 87
actions on release SFI 87
Actions Pending SFI 97
ADDBIN
SFI for 62
subcommand abbreviation 2
ADDDATASET
SFI for 62
subcommand abbreviation 2
ADDDOWNER
SFI for 62
subcommand abbreviation 2
ADDPRODUCT
SFI for 62
subcommand abbreviation 2
ADDRACK
SFI for 62
subcommand abbreviation 2
address line SFI 87
ADDVOLUME
SFI for 62
subcommand abbreviation 2
ADDVRS
SFI for 62
subcommand abbreviation 2
ADL ('803001') - Address Line 87
ADTJ ('804000') - Assigned Date 87
API methods 23
assigned date SFI 87
assigned time SFI 87
AST ('805000') - Action Status 87
ATM ('806000') - Assigned Time 87
AUD ('807000') - SMF Audit Record
Number 87
authentication 32
AVL ('808000') - Volume Availability 87

B

backup procedure name SFI 87
BDTJ ('809000') - Last Control Data Set
Backup Date 87
begin and end resource group SFIs 84
begin and end resource groups 58
BIN ('80A000') - Bin Number 87
Bin Count SFI 88
Bin Number Media Name SFI 88
bin number SFI 87
Bin Numbers in DISTANT SFI 89
Bin Numbers in LOCAL SFI 92
Bin Status SFI 98
BKPP ('80B000') - Backup Procedure
Name 87
BLKC ('80C000') - Block Count 88
BLKS ('80D000') - Block Size 88
BLKT ('80D030') - Total block count 88
block count SFI 88
block size SFI 88
BLP ('80E000') - BLP Option 88
BLP option SFI 88
BMN ('80F000') - Bin Number Media
Name 88
BTM ('810000') - Last Control Data Set
Backup Time 88

C

CACT ('811000') - Control Active
Functions 88
Catalog status SFI 89
Catalog Synchronize Date 88
Catalog Synchronize in Progress SFI 91
Catalog Synchronize Time 89
CATRETPD Retention Period SFI 88
CATS ('811800') - CATSYSID Value 88
CATSYSID Value 88
CDS ('812000') - Control Data Set
Identifier 88
CDSQ ('812900') - Control Data Set
ENQ 88
CDSU ('812100') - Control Data Set
Percentage Used SFI 88
CDTJ ('813000') - Create Date 88
CHANGEDATASET
SFI for 62
subcommand abbreviation 2
CHANGEOWNER
SFI for 62
subcommand abbreviation 2
CHANGEPRODUCT
SFI for 62
subcommand abbreviation 2
CHANGEVOLUME
SFI for 62
subcommand abbreviation 2
character set
chart xiv
use in statement xiv
CJBN ('814000') - Job Name 88
CLIB ('815000') - Current Library
Name 88
Client IP address SFI 89
Client/server host name SFI 89
closed by Abend SFI 87
CLS ('816000') - Security Class
Description 88
CMDD ('816900') - Command
Authorization - DSN SFI 88
CMDO ('8169A0') - Command
Authorization - Owner SFI 88
CNT ('817000') - Bin, Rack, or Volume
Count 88
Command Authorization - DSN SFI 88
Command Authorization - Owner
SFI 88
command classes 23
Common Time SFI 101
compound SFI 84
CONT ('057000') - Continue 86
CONT SFI 86
CONTINUE Operation 33
continuing a request 7, 78
Control Active Functions SFI 88
Control Data Set Create Date SFI 94
Control Data Set Create Time SFI 95
Control Data Set ENQ SFI 88
Control Data Set Identifier SFI 88
Control Data Set Name SFI 94
Control Data Set Percentage Used
SFI 88
Control Data Set Type SFI 95
Count of volumes stacked on a stacked
volume SFI 100
CPGM ('817820') - Creating program
name 88
Create Date SFI 88
Create Time SFI 89
Creating program name SFI 88
Creating system ID for first file SFI 90
CRP ('818000') - CATRETPD Retention
Period 88
CSDT ('818800') - Catalog Synchronize
Date 88
CSG ('819000') - Current Storage
Group 88
CSHN ('819200') - Client/server host
name 89
CSIP ('819250') - Client IP address 89
CSTM ('818800') - Catalog Synchronize
Time 89
CSVE ('819600') - Stacked volume enable
status 89
CTLG ('819800') - Catalog status 89
CTM ('81A000') - Create Time 89
CTNR ('81A300') - In container 89
Current label version SFI 93
Current Library Name SFI 88
Current Storage Group SFI 88

D

Data Check Required in IPL SFI 91
Data Class SFI 89
data format 52
Data Set Count SFI 90
Data Set Name Mask SFI 90
Data Set Name SFI 90
Data Set Recording SFI 90
Data Set Sequence SFI 90
Data Set Size SFI 90
date format 57
Date Last Referenced/Read SFI 89
Date Last Written SFI 90
DBIN ('81A600') - Destination bin number 89
DBMN ('81A700') - Destination bin media name 89
DBN ('81B000') - Bin Numbers in DISTANT 89
DC ('81C000') - Data Class 89
DD ('81D000') - DD Name 89
DD Name SFI 89
DDTJ ('81E000') - Delete Date, or Last Store Update Date 89
Default Lines Per Page SFI 92
Default Retention Period SFI 90
Delete Date SFI 89
DELETEBIN
SFI for 63
subcommand abbreviation 2
Deleted by disposition processing SFI 89
DELETEDATASET
SFI for 63
subcommand abbreviation 2
DELETEOWNER
SFI for 63
subcommand abbreviation 2
DELETEPRODUCT
SFI for 63
subcommand abbreviation 2
DELETERACK
SFI for 63
subcommand abbreviation 2
DELETEVOLUME
SFI for 63
subcommand abbreviation 2
DELETEVRS
SFI for 63
subcommand abbreviation 2
delimiters xiv
DEN ('81F000') - Media Density 89
DESC ('820000') - Volume or VRS Description 89
DEST ('821000') - Destination Name 89
Destination bin media name SFI 89
Destination bin number SFI 89
Destination Name SFI 89
Destination Type SFI 90
DEV ('822000') - Device Number 89
Device Number SFI 89
DFSMSrmm API 17
DFSMSrmm API Command C++
Classes 23
DFSMSrmm API Command Classes 23
DFSMSrmm API Command Java
Classes 23, 24
DFSMSrmm API with Web services 29

DFSMSrmm System ID SFI 99
disability 119
Disposition DD name SFI 90
Disposition Message Prefix SFI 90
DLR/DLRJ ('823000') - Date Last Referenced/Read 89
DLTD ('823700') - Deleted by disposition processing 89
DLWJ ('824000') - Date Last Written 90
DNM ('825000') - Data Set Name Mask 90
DPT ('826000') - Owner's department 90
DPT ('826000') - Owner's Department 90
DRP ('827000') - Default Retention Period 90
DSC ('828000') - Data Set Count 90
DSEQ ('829000') - Data Set Sequence 90
DSN ('82A000') - Data Set Name 90
DSPD ('82A500') - Disposition DD name 90
DSPM ('82AA00') - Disposition message prefix 90
DSR ('82B000') - Data Set Recording 90
DSS6 ('82B030') - Data Set Size 90
DSTT ('82B200') - Destination Type 90
DSYS ('82BB00') - Creating system ID for first file 90
DTE ('82C000') - Installation Date Format 90
DTM ('82D000') - Last Store Update Run Time 90

E

EBIN ('82D500') - Extended bin enable status 90
EDG_EXIT100 installation exit status 103
EDG_EXIT200 installation exit status 103
EDG_EXIT300 installation exit status 103
EDGXAPI module 3
EDGXCI macro syntax 5
EDGXCI: Call DFSMSrmm Interface 3
EDGXHINT 45
EDGXSf Structured Field Definitions 110
EML ('82DFF0') - Internet ID 90
EMN ('82E000') - Owner's Node 90
EMU ('82F000') - Owner's User ID 91
ENTN ('053000') - Number of Entries 86
ETL ('830000') - Owner's External Telephone Number 91
expanded output 55
EXPDTDROP action SFI 103
EXPDTDROP count SFI 103
EXPDTDROP percent SFI 103
Expiration Date Check SFI 103
Expiration Date Ignore SFI 102
Expiration Date SFI 103
Extended bin enable status SFI 90
External Media Type 94
External Recording Technology SFI 93

F

FCD ('831000') - Product Feature Code 91
FCSP ('831800') - Catalog Synchronize in Progress 91
FDB ('832000') - Free Bins in DISTANT Location 91
field format for data 52
FILE ('833000') - Physical File Sequence 91
FLB ('834000') - Free Bin Numbers in LOCAL 91
FOR ('835000') - Owner's Forename 91
FRB ('836000') - Free Bin Numbers in REMOTE 91
FRC ('400000') - Function Return Code 85
Free Bin Numbers in LOCAL SFI 91
Free Bin Numbers in REMOTE SFI 91
Free Bins in DISTANT Location SFI 91
Free Rack Numbers in Library SFI 91
freeing resources 42
FRK ('837000') - Free Rack Numbers in Library 91
FRS ('401000') - Function Reason Code 85
Function Reason Code SFI 85
Function Return Code SFI 85

G

GDG CYCLEBY 91
GDG DUPLICATE 91
GDGC ('837800') - GDG CYCLEBY 91
GDGD ('837805') - GDG DUPLICATE 91
Generic Rack Number SFI 91
GETVOLUME
SFI for 63
subcommand abbreviation 2
GRK ('838000') - Generic Rack Number 91

H

high level assembler 1
HLOC ('839000') - Home Location 91
HLOT ('839200') - Home Location Type 91
Home Location SFI 91
Home Location Type SFI 91

I

In container SFI 89
Input action SFI 96
Input ignore condition SFI 96
Input reject condition SFI 96
Installation Date Format SFI 90
Installation RACF Support SFI 98
Integrated Removable Media Manager SFI 91
Internet ID SFI 90
INTR ('83A000') - Volume Intransit Status 91

IPL ('83B000') - Data Check Required in IPL 91
IRMM ('83B30') - Integrated Removable Media Manager 91
ITL ('83C000') - Owner's Internal Telephone Number 91

J

JBDT ('83CA00') - Last Journal Backup Date SFI 91
JBTM ('83CB00') - Last Journal Backup Time SFI 91
JDS ('83D000') - Journal Name 92
Job Name SFI 88
Journal Name SFI 92
Journal Percentage Used SFI 92
Journal status SFI 92
Journal transaction SFI 92
JOURNALFULL Parmlib Value SFI 92
JRNF ('83E000') - JOURNALFULL Parmlib Value 92
JRNS ('83EA00') - Journal status 92
JRNT ('83ED00') - Journal transaction 92
JRNU ('83F000') - Journal Percentage Used 92

K

KEL1 ('83F500') - Key encryption key label 1 92
KEL2 ('83F505') - Key encryption key label 2 92
KEM1 ('83F520') - Key encoding mechanism for key label 1 92
KEM2 ('83F525') - Key encoding mechanism for key label 2 92
Key encoding mechanism for key label 1 SFI 92
Key encoding mechanism for key label 2 SFI 92
Key encryption key label 1 SFI 92
Key encryption key label 2 SFI 92
Key From SFI 86
Key to SFI 86
keyboard 119
KEYF ('054000') - Key From 86
KEYT ('054200') - Key to 86

L

Last Change User ID SFI 92
Last Control Data Set Backup Date SFI 87
Last Control Data Set Backup Time SFI 88
Last Control Data Set Extract Date SFI 98
Last Control Data Set Extract Time SFI 99
Last Drive SFI 92
Last Expiration Processing Start Date SFI 99
Last Expiration Processing Start Time SFI 99

Last Inventory Management Expiration Time SFI 103
Last Inventory Management Processing Date SFI 101
Last Inventory Management VRS Time SFI 102
Last Journal Backup Date SFI 91
Last Journal Backup Time SFI 91
Last Store Update Date SFI 89
Last Store Update Run Time SFI 90
Last used DD name SFI 92
Last used job name SFI 93
Last used program name SFI 93
Last used step name SFI 93
LBL ('840000') - Volume Label Type 92
LBN ('841000') - Bin Numbers in LOCAL 92
LCID ('842000') - Last Change User ID 92
LCT ('843000') - Default Lines Per Page 92
LCTK ('843100') - Local tasks 92
LDD ('843B00') - Last used DD name 92
LDDF ('844000') - Location Definition Exists 92
LDEV ('845000') - Last Drive 92
LDLC ('846000') - Location Name 92
LDLT ('847000') - Location Type 92
LDMN ('848000') - Location Media Name 92
LDMT ('849000') - Location Management Type 93
LDPR ('84A000') - Location Priority 93
Library Rack Numbers SFI 93
limiting the amount of information returned 78
LINE ('84B000') - Output Data Line 93
line format for data 52
LISTBIN
SFI for 63
subcommand abbreviation 2
LISTCONTROL
SFI for 64
subcommand abbreviation 2
LISTDATASET
SFI for 68
subcommand abbreviation 2
LISTOWNER
SFI for 69
subcommand abbreviation 2
LISTPRODUCT
SFI for 70
subcommand abbreviation 2
LISTRACK
SFI for 70
subcommand abbreviation 2
LISTVOLUME
SFI for 71
subcommand abbreviation 2
LISTVRS
SFI for 73
subcommand abbreviation 2
LJOB ('84B420') - Last used job name 93
LOAN ('84C000') - Loan Location 93
Loan Location SFI 93
LOC ('84D000') - Location 93
Local tasks SFI 92

Location Definition Exists SFI 92
Location Management Type SFI 93
Location Media Name SFI 92
Location name SFI 97
Location Name SFI 92
Location Priority SFI 93
Location SFI 93
Location Type SFI 92, 93
LOCT ('84E000') - Location Type 93
Logical Record Length SFI 93
LookAt message retrieval tool x
LPGM ('84E760') - Last used program name 93
LRCL ('84F000') - Logical Record Length 93
LRK ('850000') - Number of Library Rack Numbers 93
LSTP ('850370') - Last used step name 93
LVC ('850500') - current label version 93
LVN ('850A00') - Required label version 93

M

Management Class SFI 93
mapping macros
EDGXCI 109
EDGXSF 110
Master Overwrite SFI 94
Matching VRS Job Name SFI 101
Matching VRS Name SFI 102
Matching VRS Type SFI 103
MAXHOLD Value SFI 99
Maximum Retention Period SFI 94
MC ('851000') - Management Class 93
MDNF ('851400') - Media Information Name 93
MDRA ('851980') - MEDINF replace policy 93
MDRP ('8519C0') - MEDINF replace policy 93
MDRT ('8519E0') - MEDINF replace policy 93
MDRW ('8519F0') - MEDINF replace policy 93
MDRX ('851A00') - External Recording Technology 93
MDS ('852000') - Control Data Set Name 94
MDTJ ('853000') - Control Data Set Create Date 94
MDTX ('853400') - External Media Type 94
MEDA ('854000') - Media Special Attributes 94
MEDC ('855000') - Media Compaction 94
Media Compaction SFI 94
Media Density SFI 89
Media Information Name SFI 93
Media Name SFI 94
Media Recording Format SFI 94
Media Special Attributes SFI 94
Media Type SFI 94
MEDINF replace policy SFI 93
MEDN ('856000') - Media Name 94

MEDR ('857000') - Media Recording Format 94
 MEDT ('858000') - Media Type 94
 memory size limitation 30
 Message Line SFI 86
 Message Number SFI 86
 message retrieval tool, LookAt x
 Message SFIs 86
 Message Text Case SFI 95
 Message Variable SFIs 86
 MFR ('859000') - Source Location Name 94
 MID ('85A000') - Mount message ID 94
 MIV ('85A500') - Moving-in volume 94
 MOP ('85C000') - Master Overwrite 94
 Mount message ID SFI 94
 MOV ('85A900') - Moving-out volume 94
 Move By SFI 95
 Move Mode SFI 94
 Move Status SFI 95
 Move Type SFI 95
 Movement Tracking Date SFI 99
 Moving-in volume SFI 94
 Moving-out volume SFI 94
 MOVMT ('85B000') - Move Mode 94
 MRP ('85D000') - Maximum Retention Period 94
 MSGF ('85E000') - Case of Message Text 95
 MSGL ('051000') - Message Line 86
 MSGN ('052000') - Message Number 86
 MST ('85F000') - Move Status 95
 MTM ('860000') - Control Data Set Create Time 95
 MTO ('861000') - Target Location Name 95
 MTP ('862000') - Control Data Set Type 95
 MTY ('862800') - Move Type 95
 multiple parameter list, multiple token areas 40
 multiple parameter list, single token area 40
 MVBY ('862B00') - Move By 95
 MVS ('863000') - MVS Use 95
 MVS Use SFI 95

N

Next Vital Record Specification Name SFI 95
 Next Volume SFI 95
 Next VRS Value SFI 101
 NLOC ('865000') - Required Location 95
 NLOT ('865200') - Required location type 95
 NME ('866000') - Security Class Name 95
 NOSMT action for partition entry SFI 97
 NOT ('866800') - Notify 95
 Number of Bin Numbers in REMOTE SFI 98
 Number of Entries SFI 86
 Number of Volumes SFI 101
 NVL ('867000') - Next Volume 95
 NVRS ('868000') - Next VRS Name 95

O

OAC ('869000') - Owner Access 95
 OBMN ('86A000') - Old Bin Number Media Name 95
 OBN ('86B000') - Old Bin Number 95
 obtaining space for output buffer 13
 OCE ('86B800') - Volume Information Recorded at O/C/EOV 95
 Offset to Message ID SFI 99
 Old Bin Number Media Name SFI 95
 Old Bin Number SFI 95
 Old Location SFI 95
 Old location type SFI 96
 Old volume SFI 96
 OLOC ('86C000') - Old Location 95
 OLOT ('86C200') - Old location type 96
 Operating Mode SFI 96
 OPL ('86D000') - Position of Rack Number or Pool ID 96
 OPM ('86E000') - Operating Mode 96
 ORIA ('86E8A0') - Input action 96
 Original Expiration Date SFI 96
 ORII ('86E8A8') - Input ignore condition 96
 ORIR ('86E8B8') - Input reject condition 96
 OROA ('86EA00') - Output action 96
 OROI ('86EA08') - Output ignore condition 96
 OROR ('86EA18') - Output reject condition 96
 ORTP ('86EF08') - Type of open rule entry 96
 ORVE ('86EF8F') - Volume range end 96
 ORVL ('86EF85') - Volume serial number 96
 ORVS ('86EF80') - Volume range start 96
 Output action SFI 96
 output buffer
 hexadecimal example of an output buffer 115
 obtaining space for 13
 Output Data Line SFI 93
 Output ignore condition SFI 96
 Output reject condition SFI 96
 OVL ('86F000') - Position of Volume Serial 96
 OVOL ('86F500') - Old volume 96
 OWN ('870000') - Owner 96
 Owner Access SFI 95
 Owner SFI 96
 Owner's department SFI 90
 Owner's Department SFI 90
 Owner's External Telephone Number SFI 91
 Owner's Forename SFI 91
 Owner's Internal Telephone Number SFI 91
 Owner's Node SFI 90
 Owner's Surname SFI 100
 Owner's User ID SFI 91
 OXDJ ('871000') - Original Expiration Date 96

P

PACS ('801800') - PREACS 97
 parallel processing 32
 parameter lists
 multiple parameter list, multiple token areas 40
 multiple parameter list, single token area 40
 single parameter list, multiple token areas 38
 single parameter list, single token area 36
 Parmlib Member Suffix SFI 97
 PDA ('871E00') - PDA state 97
 PDA block count SFI 97
 PDA block size SFI 97
 PDA log state SFI 97
 PDA state SFI 97
 PDAC ('871E90') - PDA block count 97
 PDAL ('871E30') - PDA log state 97
 PDAS ('871E90') - PDA block size 97
 PDS ('872000') - Pool Description 97
 PDSC ('873000') - Product Description 97
 PEND ('874000') - Actions Pending 97
 Permanent Read Error SFI 97
 Permanent Write Error SFI 98
 persistence processing 32
 Physical File Sequence SFI 91
 PID ('875000') - Pool Prefix 97
 PLN ('876000') - Pool Name 97
 PNME ('877000') - Product Software Name 97
 PNUM ('878000') - Software Product Number 97
 Pool Definition Pool Type SFI 97
 Pool Definition RACF Option SFI 97
 Pool Definition System ID SFI 97
 Pool Description SFI 97
 Pool Name SFI 97
 Pool Prefix SFI 97
 Position of Rack Number or Pool ID SFI 96
 Position of Volume Serial SFI 96
 PRD ('879000') - Permanent Read Errors 97
 PREACS SFI 97
 Previous Volume SFI 98
 PRF ('87A000') - Pool Definition RACF Option 97
 Primary VRS Subchain Name SFI 102
 Primary VRS Subchain Start Date SFI 102
 Priority SFI 97
 Product Description SFI 97
 Product Feature Code SFI 91
 Product Software Name SFI 97
 Programming Guidelines 33
 programming requirements 3
 PRTY ('87B000') - Priority 97
 PSF2 ('87C010') - Second Parmlib Member Suffix 97
 PSFX ('87C000') - Parmlib Member Suffix 97
 PSN ('87D000') - Pool Definition System ID 97

PTNA ('87DB00') - NOSMT action for partition entry 97
 PTNL ('87DB0C') - Location name 97
 PTP ('87E000') - Pool Definition Pool Type 97
 PTSA ('87EB80') - SMT action for partition entry 98
 PTPP ('87EBA8') - Type of partition entry 98
 PTVE ('87EC0F') - Volume range end 98
 PTVL ('87EC08') - Volume serial number 98
 PTVS ('87EC00') - Volume range start 98
 PVL ('87F000') - Previous Volume 98
 PWT ('880000') - Permanent Write Errors 98

R

Rack Count SFI 88
 Rack Number or Bin Number SFI 98
 Rack Status SFI 98
 RBN ('881000') - Number of Bin Numbers in REMOTE 98
 RBYS ('881200') - Retain by set 98
 RCF ('882000') - Installation RACF Support 98
 RCFM ('883000') - Record Format 98
 RCK ('884000') - Rack Number or Bin Number 98
 RDTJ ('886000') - Last Control Data Set Extract Date 98
 Reason Code SFI 85
 Reason code SFIs 85
 Record Format SFI 98
 Reject Type SFI 100
 Release Action Scratch Immediate SFI 102
 releasing all resources 39
 Required label version SFI 93
 Required Location SFI 95
 Required location type SFI 95 resources
 freeing 42
 obtaining 33
 releasing 43
 RET ('888000') - Retention Type 98
 Retain by set SFI 98
 Retain by SFI 98
 Retention Date SFI 98
 Retention Type SFI 98
 Return Code SFI 85
 Return Code SFIs 85
 Reuse bin at SFI 99
 reusing resources 33
 RSNC ('402000') - Reason Code 85
 RST ('88A000') - Rack or Bin Status 98
 RTBY ('88B900') - Retain by 98
 RTDJ ('88C000') - Retention Date 98
 RTM ('88E000') - Last Control Data Set Extract Time 99
 RTNC ('403000') - Return Code 85
 RUB ('88E500') - Reuse bin at 99

S

SC ('890000') - Storage Class 99
 SC1 ('894000') - Storenumber 99
 Scratch Immediate SFI 102
 Scratch mode SFI 99
 Scratch Procedure Name SFI 99
 SCRML ('891000') - Scratch mode 99
 SCST ('892000') - Security Class Status 99
 SDTJ ('895000') - Movement Tracking Date 99
 SEARCHBIN
 SFIs for 74
 subcommand abbreviation 2
 SEARCHDATASET
 SFIs for 75
 subcommand abbreviation 2
 SEARCHOWNER
 SFIs for 75
 SEARCHPRODUCT
 SFIs for 76
 subcommand abbreviation 2
 SEARCHRACK
 limiting the amount of information returned 78
 SFIs for 76
 subcommand abbreviation 2
 SEARCHVOLUME
 SFIs for 76
 subcommand abbreviation 2
 SEARCHVRS
 SFIs for 77
 subcommand abbreviation 2
 SEC ('896000') - Security Class Number 99
 Second Parmlib Member Suffix SFI 97
 Secondary VRS Jobname Mask SFI 103
 Secondary VRS Mask SFI 103
 Secondary VRS Subchain Name SFI 104
 Secondary VRS Subchain Start Date SFI 104
 Security Class Description SFI 88
 Security Class Name SFI 95
 Security Class Number SFI 99
 Security Class Status SFI 99
 SEQ ('898000') - Volume Sequence 99
 Server host name SFI 99
 Server IP address SFI 99
 Server number SFI 99
 Server tasks SFI 100
 Service Name SFI 86
 SG ('89A000') - Storage Group Name 99
 shortcut keys 119
 SID ('89B000') - DFSMSrmm System ID 99
 single parameter list, multiple token areas 38
 single parameter list, single token area 36
 SLM ('89C000') - MAXHOLD Value 99
 SMF audit record number SFI 87
 SMF Security Record Number SFI 100
 SMF System ID SFI 100
 SMI ('89E000') - Offset to Message ID 99
 SMP ('89E210') - System-managed tape purge 99
 SMSACS SFI 87

SMT action for partition entry SFI 98
 SMU ('89E220') - System-managed tape update 99
 Software Product Number SFI 97
 Software Product Version SFI 101
 software requirements 1
 SOSJ ('89F000') - Last Expiration Processing Start Date 99
 SOSP ('8A0000') - Scratch Procedure Name 99
 SOST ('8A1000') - Last XPROC Start Time 99
 Source Location Name SFI 94
 SRHN ('8A1A00') - Server host name 99
 SRIP ('8A1A30') - Server IP address 99
 SRIP ('8A2500') - Subsystem type 100
 SRPN ('8A1A50') - Server number 99
 SRTK ('8A1AF0') - Server tasks 100
 SSM ('8A2000') - SMF Security Record Number 100
 Stacked volume enable status SFI 89
 standard output 55
 STEP ('8A3000') - Step Name 100
 Step Name SFI 100
 Storage Class SFI 99
 Storage Group Name SFI 99
 Storenumber SFI 99
 Structured Field Introducer
 data format 52
 definitions of 83
 for begin and end resource groups 84
 for Messages and Message Variables 86
 for Return and Reason Codes 85
 for subcommand output data 86
 format 83
 STVC ('8A3800') - Count of volumes stacked on a stacked volume 100
 subcommand output data SFIs 86
 Subsystem type SFI 100
 supported subcommands 2
 SUR ('8A4000') - Owner's Surname 100
 SVCN ('404000') - Service Name 86
 syntax for EDGXCI 5
 SYS ('8A5000') - SMF System ID 100
 System-managed tape purge SFI 99
 System-managed tape update SFI 99

T

TAC ('8A6000') - Reject Type 100
 Tape volume exit purge option SFI 100
 Target Location Name SFI 95
 Temporary Read Error SFI 100
 Temporary Write Error SFI 100
 time format 57
 Time Last Referenced SFI 100
 Time Zone SFI 100
 time zones
 using different 58
 TLR ('8A6800') - Time Last Referenced 100
 Total block count SFI 88
 TRD ('8A7000') - Temporary Read Errors 100

TVXP ('8A7900') - Tape volume exit
 purge option 100
 TWT ('8A8000') - Temporary Write
 Errors 100
 TYP ('8A9000') - VRS Type 100
 TYPE ('055200') - Type To 86
 Type From SFI 86
 Type of open rule entry SFI 96
 Type of partition entry SFI 98
 Type To SFI 86
 Types of Structured Field Introducers 58
 TYPF ('055000') - Type From 86
 TZ ('8A9E00') - Time Zone 100
 TZ SFI 58

U

UDDI registry 31
 UDTJ ('8AA000') - User ID 100
 UID ('8AB001') - User ID 100
 UNC ('8AC000') - Uncatalog Option 100
 Uncatalog Option SFI 100
 USE6 ('8AE030') - Volume Usage 101
 USEC ('8AD000') - Volume Use
 Count 101
 USEM ('8AE000') - Volume Usage
 (KB) 101
 User ID SFI 100, 101
 User Notification SFI 95
 using multiple parameter lists 35
 UTC ('8AE600') - Common Time 101
 UTM ('8AE800') - User ID 101

V

VAC ('8AF001') - Volume Access 101
 VACT ('8B0000') - VRSMIN Action 101
 VANX ('8B0800') - Next VRS Value 101
 VCAP ('8B0B00') - Volume/Media
 capacity 101
 VCHG ('8B1000') - VRSCCHANGE
 Value 101
 VDD ('8B2000') - VRS Delay Days 101
 VDRA ('8B2800') - VRSDROP action 101
 VDRC ('8B2802') - VRSDROP count 101
 VDRP ('8B280F') - VRSDROP
 percent 101
 VDTJ ('8B3000') - Last Inventory
 Management Processing Date 101
 Vendor information SFI 102
 VER ('8B4000') - Software Product
 Version 101
 Vital Record Count SFI 102
 Vital Record Specification Delay Days
 SFI 101
 Vital record specification name SFI 102
 Vital Record Specification SFI 97
 Vital Record Specification Type SFI 100
 VJBN ('8B5000') - Matching VRS Job
 Name 101
 VLN ('8B6000') - Number of
 Volumes 101
 VM ('8B7000') - VM Use 101
 VM Use SFI 101
 VMIN ('8B8000') - VRSMIN Count
 Value 102

VMV ('8B9000') - VRS Management
 Value 102
 VNDR ('8B9E00') - Vendor
 information 102
 VNME ('8BA000') - Matching VRS
 Name 102
 VOL ('8BC000') - Volume Serial 102
 VOL1 ('8BCD00') - VOL1 label
 volser 102
 VOL1 label volser SFI 102
 VOLT ('8BC200') - Volume type 102
 Volume Access SFI 101
 volume availability SFI 87
 Volume Count SFI 88
 Volume Description SFI 89
 Volume Information Recorded at
 O/C/EVO Indicator SFI 95
 Volume Intransit Status SFI 91
 Volume Label Type SFI 92
 Volume percent full SFI 102
 Volume range end SFI 96, 98
 Volume range start SFI 96, 98
 Volume Sequence SFI 99
 Volume serial number SFI 96, 98
 Volume Serial SFI 102
 Volume Status SFI 102
 Volume type SFI 102
 Volume Usage SFI 101
 Volume Use Count SFI 101
 Volume write mount count SFI 102
 Volume/Media capacity 101
 VPCT ('8BC300') - Volume percent
 full 102
 VRC ('8BD000') - Vital Record
 Count 102
 VREA ('8BD500') - VRSRETAIN
 action 102
 VREC ('8BD502') - VRSRETAIN
 count 102
 VREP ('8BD50F') - VRSRETAIN
 percent 102
 VRJ ('8BE000') - VRS Job Name 102
 VRS ('8BF000') - Vital record specification
 name 102
 VRS Description SFI 89
 VRS Job Name SFI 101, 102
 VRS Management Value SFI 102
 VRS Retained Status SFI 102
 VRSCCHANGE Value SFI 101
 VRSDROP action SFI 101
 VRSDROP count SFI 101
 VRSDROP percent SFI 101
 VRSEL Value SFI 102
 VRSI ('8BF500') - Scratch immediate 102
 VRSL ('8BFA00') - VRSEL Value 102
 VRSMIN Action SFI 101
 VRSMIN Count Value SFI 102
 VRSR ('8C0000') - VRS Retained
 Status 102
 VRSRETAIN action SFI 102
 VRSRETAIN count SFI 102
 VRSRETAIN percent SFI 102
 VRXI ('8C0800') - Expiration Date
 Ignore 102
 VSCD ('8C1000') - Primary VRS Subchain
 Start Date 102

VSCN ('8C1800') - Primary VRS Subchain
 Name 102
 VST ('8C2000') - Volume Status 102
 VTM ('8C3000') - Last Inventory
 Management VRS Time 102
 VTYP ('8C4000') - Matching VRS
 Type 103
 VWMC ('8B9100') - Volume write mount
 count 102

W

Web service client
 sample 31
 World-wide identifier SFI 103
 WORM ('8C4300') - Write Once Read
 Many 103
 Write Once Read Many SFI 103
 WWID ('8C4500') - World-wide
 identifier 103

X

X100 ('8C78020') - EDG_EXIT100
 installation exit status 103
 X200 ('8C7801') - EDG_EXIT200
 installation exit status 103
 X300 ('8C7802') - EDG_EXIT300
 installation exit status 103
 XDC ('8C5000') - Expiration Date
 Check 103
 XDRA ('8C5D00') - EXPDTPDROP
 action 103
 XDRC ('8C5D02') - EXPDTPDROP
 count 103
 XDRP ('8C5D0F') - EXPDTPDROP
 percent 103
 XDTJ ('8C6000') - Expiration Date 103
 XML output 25, 29
 XTM ('8C7000') - Last Inventory
 Management Expiration Time 103

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

z/OS
DFSMSrmm Application Programming Interface

Publication No. SC26-7403-09

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

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. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send your comments via e-mail to: mhvrcfs@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address



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

IBM Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY
12601-5400



Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5694-A01

Printed in USA

SC26-7403-09

