DFSMSrmm Application Programming Interface
DFSMSrmm Application Programming Interface
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 ............... xii
How to abbreviate commands and operands .. 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 ................................ xvi
Changed Information ......................... xvi
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 ............................... xvi
New Information ................................ xvi
Changed Information ......................... xvi
Summary of Changes for SC26-7403-06 z/OS
Version 1 Release 8 ............................... xvi
New Information ................................ xvi
Changed Information ......................... xvi

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
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requesting Field Format</td>
<td>53</td>
</tr>
<tr>
<td>Requesting Types of Output</td>
<td>55</td>
</tr>
<tr>
<td>Requesting Standard Output</td>
<td>55</td>
</tr>
<tr>
<td>Requesting Expanded Output</td>
<td>55</td>
</tr>
<tr>
<td>Accessing Return and Reason Codes</td>
<td>56</td>
</tr>
<tr>
<td>Accessing Messages and Message Variables</td>
<td>57</td>
</tr>
<tr>
<td>Interpreting Date Format and Time Format</td>
<td>57</td>
</tr>
<tr>
<td>Using Different Time Zones</td>
<td>58</td>
</tr>
<tr>
<td>Identifying Structured Field Introducers</td>
<td>58</td>
</tr>
<tr>
<td>Begin and End Resource Groups</td>
<td>58</td>
</tr>
<tr>
<td>System Return and Reason Code SFIs</td>
<td>59</td>
</tr>
<tr>
<td>Messages and Message Variables SFIs</td>
<td>60</td>
</tr>
<tr>
<td>SFIs for Output Data for Subcommands</td>
<td>61</td>
</tr>
<tr>
<td>ADD-Type of Subcommands</td>
<td>62</td>
</tr>
<tr>
<td>CHANGE-Type of Subcommands</td>
<td>62</td>
</tr>
<tr>
<td>DELETE-Type of Subcommands</td>
<td>63</td>
</tr>
<tr>
<td>GETVOLUME Subcommand</td>
<td>63</td>
</tr>
<tr>
<td>LIST-Type of Subcommands</td>
<td>63</td>
</tr>
<tr>
<td>LISTBIN SFIs</td>
<td>63</td>
</tr>
<tr>
<td>LISTCONTROL SFIs</td>
<td>64</td>
</tr>
<tr>
<td>LISTDATASET SFIs</td>
<td>68</td>
</tr>
<tr>
<td>LISTOWNER SFIs</td>
<td>69</td>
</tr>
<tr>
<td>LISTPRODUCT SFIs</td>
<td>70</td>
</tr>
<tr>
<td>LISTTRACK SFIs</td>
<td>70</td>
</tr>
<tr>
<td>LISTVOLUME SFIs</td>
<td>71</td>
</tr>
<tr>
<td>LISTVRS SFIs</td>
<td>73</td>
</tr>
<tr>
<td>SEARCH-Type of Subcommands</td>
<td>74</td>
</tr>
<tr>
<td>SEARCHBIN SFIs</td>
<td>74</td>
</tr>
<tr>
<td>SEARCHDATASET SFIs</td>
<td>75</td>
</tr>
<tr>
<td>SEARCHOWNER SFIs</td>
<td>75</td>
</tr>
<tr>
<td>SEARCHPRODUCT SFIs</td>
<td>76</td>
</tr>
<tr>
<td>SEARCHRACK SFIs</td>
<td>76</td>
</tr>
<tr>
<td>SEARCHVOLUME SFIs</td>
<td>76</td>
</tr>
<tr>
<td>SEARCHVRS SFIs</td>
<td>77</td>
</tr>
<tr>
<td>Controlling Output from List and Search Type Requests</td>
<td>78</td>
</tr>
<tr>
<td>Limiting the Search for a Request</td>
<td>78</td>
</tr>
<tr>
<td>Output Buffer Examples</td>
<td>78</td>
</tr>
<tr>
<td>Appendix A. Structured Field Introducers</td>
<td>83</td>
</tr>
<tr>
<td>SFI Format</td>
<td>83</td>
</tr>
<tr>
<td>Structured Field Lengths</td>
<td>83</td>
</tr>
<tr>
<td>Compound SFI</td>
<td>84</td>
</tr>
<tr>
<td>SFIs for Begin and End Resource Groups</td>
<td>84</td>
</tr>
<tr>
<td>SFIs for Return and Reason Codes</td>
<td>85</td>
</tr>
<tr>
<td>SFIs for Messages and Message Variables</td>
<td>86</td>
</tr>
<tr>
<td>SFIs for Subcommand Output Data</td>
<td>86</td>
</tr>
<tr>
<td>Appendix B. Structured Field Introducers by Subcommand</td>
<td>105</td>
</tr>
<tr>
<td>Appendix C. DFSMSrmm Application Programming Interface Mapping</td>
<td>109</td>
</tr>
<tr>
<td>Macros</td>
<td>109</td>
</tr>
<tr>
<td>EDGXCI: Parameter List</td>
<td>109</td>
</tr>
<tr>
<td>EDGXSF: Structured Field Definitions</td>
<td>110</td>
</tr>
<tr>
<td>EDGXSF Parameters</td>
<td>110</td>
</tr>
<tr>
<td>EDGXSF Mapping</td>
<td>111</td>
</tr>
<tr>
<td>EDGXSF Labeling Conventions</td>
<td>112</td>
</tr>
<tr>
<td>Appendix D. Hexadecimal Example of an Output Buffer</td>
<td>115</td>
</tr>
<tr>
<td>Hexadecimal Representation of an Output Buffer</td>
<td>115</td>
</tr>
<tr>
<td>Description of the Contents of an Output Buffer</td>
<td>115</td>
</tr>
<tr>
<td>Processing the Contents of an Output Buffer</td>
<td>117</td>
</tr>
<tr>
<td>Appendix E. Accessibility</td>
<td>119</td>
</tr>
<tr>
<td>Using assistive technologies</td>
<td>119</td>
</tr>
<tr>
<td>Keyboard navigation of the user interface</td>
<td>119</td>
</tr>
<tr>
<td>z/OS information</td>
<td>119</td>
</tr>
<tr>
<td>Notices</td>
<td>121</td>
</tr>
<tr>
<td>Programming interface information</td>
<td>122</td>
</tr>
<tr>
<td>Policy for unsupported hardware</td>
<td>122</td>
</tr>
<tr>
<td>Trademarks</td>
<td>122</td>
</tr>
<tr>
<td>Index</td>
<td>123</td>
</tr>
</tbody>
</table>
Figures

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

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

<table>
<thead>
<tr>
<th>Publication Title</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS DFSMSrmm Diagnosis Guide</td>
<td>GY27-7619</td>
</tr>
<tr>
<td>z/OS DFSMSrmm Managing and Using Removable Media</td>
<td>SC26-7404</td>
</tr>
<tr>
<td>z/OS DFSMSrmm Implementation and Customization Guide</td>
<td>SC26-7405</td>
</tr>
<tr>
<td>z/OS DFSMSrmm Reporting</td>
<td>SC26-7406</td>
</tr>
</tbody>
</table>

This document also refers to the following publications:

<table>
<thead>
<tr>
<th>Publication Title</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS XL C/C++ User’s Guide</td>
<td>SC09-4767</td>
</tr>
<tr>
<td>z/OS Migration</td>
<td>GA22-7499</td>
</tr>
</tbody>
</table>
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.

\[ \text{Syntax Diagram} \]

- 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.

\[ \text{LISTDATASET} \rightarrow \text{data_set_name} \rightarrow \text{VOLUME(volume_serial)} \]

\[ \text{FILESEQ} \rightarrow \text{physical_file_sequence_number} \]

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

\[ \text{REQUIRED_KEYWORD} \]

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

\[ \text{REQUIRED_KEYWORD_OR_VALUE_1} \rightarrow \text{REQUIRED_KEYWORD_OR_VALUE_2} \]

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

\[ \text{KEYWORD} \]

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

\[ \text{KEYWORD_OR_VALUE_1} \rightarrow \text{KEYWORD_OR_VALUE_2} \]
• 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.

```
  REPEATABLE_KEYWORD
```

• 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.

```
  REPEATABLE_KEYWORD OR VALUE_1
  REPEATABLE_KEYWORD OR VALUE_2
```

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

```
  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)*.

```
  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)*.

```
  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.

```
  DEFAULT
```

• 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.

```
  (1)
```

**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:

```
1ST_KEYWORD,2ND_KEYWORD,3RD_KEYWORD
```

The following is an example of a syntax diagram.

```
DELETEOWNER owner_ID 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

<table>
<thead>
<tr>
<th>Character Set</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphanumeric</td>
<td>Alphabetic</td>
</tr>
<tr>
<td></td>
<td>Numeric</td>
</tr>
<tr>
<td>National (See note)</td>
<td>“At” sign</td>
</tr>
<tr>
<td></td>
<td>Dollar sign</td>
</tr>
<tr>
<td></td>
<td>Pound sign</td>
</tr>
<tr>
<td>Special</td>
<td>Comma</td>
</tr>
<tr>
<td></td>
<td>Period</td>
</tr>
<tr>
<td></td>
<td>Slash</td>
</tr>
<tr>
<td></td>
<td>Apostrophe</td>
</tr>
<tr>
<td></td>
<td>Left parenthesis</td>
</tr>
<tr>
<td></td>
<td>Right parenthesis</td>
</tr>
<tr>
<td></td>
<td>Asterisk</td>
</tr>
<tr>
<td></td>
<td>Ampersand</td>
</tr>
<tr>
<td></td>
<td>Plus sign</td>
</tr>
<tr>
<td></td>
<td>Hyphen</td>
</tr>
<tr>
<td></td>
<td>Equal sign</td>
</tr>
<tr>
<td></td>
<td>Blank</td>
</tr>
<tr>
<td>EBCDIC text</td>
<td>EBCDIC printable character set</td>
</tr>
<tr>
<td></td>
<td>Characters that can be represented by hexadecimal X'40' through X'FE'</td>
</tr>
</tbody>
</table>

Note: The system recognizes the following hexadecimal representations of the U.S. National characters; @ as X'7C'; $ as X'3B'; and # as X'7B'. In countries other than the U.S., the U.S. National characters represented on terminal keyboards might generate a different hexadecimal representation and cause an error. For example, in some countries the $ character may generate a X'4A'.

Table 2. Special Characters Used in Syntax

<table>
<thead>
<tr>
<th>Character</th>
<th>Syntactical Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>To separate parameters and subparameters</td>
</tr>
<tr>
<td>=</td>
<td>To separate a keyword from its value, for example, BURST=YES</td>
</tr>
<tr>
<td>( b )</td>
<td>To enclose subparameter list or the member name of a PDS or PDSE</td>
</tr>
<tr>
<td>&amp;</td>
<td>To identify a symbolic parameter, for example, &amp;LIB</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>To identify a temporary data set name, for example, &amp;&amp;TEMPDS, and, to identify an in-stream or sysout data set name, for example, &amp;&amp;PAYOUT</td>
</tr>
<tr>
<td>.</td>
<td>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</td>
</tr>
<tr>
<td>*</td>
<td>To refer to an earlier statement, for example, OUTPUT=<em>.name, or, in certain statements, to indicate special functions: //label CNTL * //ddname DD * RESTART=</em> on the JOB statement</td>
</tr>
<tr>
<td>'</td>
<td>To enclose specified parameter values which contain special characters</td>
</tr>
<tr>
<td>(blank)</td>
<td>To delimit fields</td>
</tr>
</tbody>
</table>
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
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 EXPDTDROP action.
• XDRC for EXPDTDROP count.
• XDRP for EXPDTDROP percent.
• X300 for UX300 installation exit status.
• Many new SFIs for OPENRULE and PRTRITION.

Changed Information


Updated the following structured field introducers:
• AUD for SMF audit record type.
• MEDR for recording technology.
• MEDI 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>
<thead>
<tr>
<th>Group</th>
<th>Subcommand</th>
<th>Abbrev</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>ADDBIN</td>
<td>AB</td>
<td>Add bin number information</td>
</tr>
<tr>
<td></td>
<td>ADDDDATASET</td>
<td>AD</td>
<td>Add data set information</td>
</tr>
<tr>
<td></td>
<td>ADDOWNER</td>
<td>AO</td>
<td>Add owner information</td>
</tr>
<tr>
<td></td>
<td>ADDPRODUCT</td>
<td>AP</td>
<td>Add software product information</td>
</tr>
<tr>
<td></td>
<td>ADDRACK</td>
<td>AR</td>
<td>Add shelf location information</td>
</tr>
<tr>
<td></td>
<td>ADDVOLUME</td>
<td>AV</td>
<td>Add volume information</td>
</tr>
<tr>
<td></td>
<td>ADDVRS</td>
<td>AS</td>
<td>Add a vital record specification</td>
</tr>
<tr>
<td>Change</td>
<td>CHANGEDATASET</td>
<td>CD</td>
<td>Change data set information</td>
</tr>
<tr>
<td></td>
<td>CHANGEOWNER</td>
<td>CO</td>
<td>Change owner information</td>
</tr>
<tr>
<td></td>
<td>CHANGEPRODUCT</td>
<td>CP</td>
<td>Change software product information</td>
</tr>
<tr>
<td></td>
<td>CHANGEOVOLUME</td>
<td>CV</td>
<td>Change volume information</td>
</tr>
<tr>
<td>Delete</td>
<td>DELETEBIN</td>
<td>DB</td>
<td>Delete bin number information</td>
</tr>
<tr>
<td></td>
<td>DELETEDATASET</td>
<td>DD</td>
<td>Delete data set information</td>
</tr>
<tr>
<td></td>
<td>DELETEOWNER</td>
<td>DO</td>
<td>Delete owner information</td>
</tr>
<tr>
<td></td>
<td>DELETERACK</td>
<td>DR</td>
<td>Delete shelf location information</td>
</tr>
<tr>
<td></td>
<td>DELETEVOLUME</td>
<td>DV</td>
<td>Release a volume and delete volume</td>
</tr>
<tr>
<td></td>
<td>DELETEVRS</td>
<td>DS</td>
<td>Delete a vital record specification</td>
</tr>
<tr>
<td>Get</td>
<td>GETVOLUME</td>
<td>GV</td>
<td>Request or assign a volume</td>
</tr>
<tr>
<td>List</td>
<td>LISTBIN</td>
<td>LB</td>
<td>Display bin number information</td>
</tr>
<tr>
<td></td>
<td>LISTCONTROL</td>
<td>LC</td>
<td>Display PARMLIB options and control information</td>
</tr>
<tr>
<td></td>
<td>LISTDATASET</td>
<td>LD</td>
<td>Display data set information</td>
</tr>
<tr>
<td></td>
<td>LISTOWNER</td>
<td>LO</td>
<td>Display owner information</td>
</tr>
<tr>
<td></td>
<td>LISTPRODUCT</td>
<td>LP</td>
<td>Display software product information</td>
</tr>
<tr>
<td></td>
<td>LISTRACK</td>
<td>LR</td>
<td>Display shelf location information</td>
</tr>
<tr>
<td></td>
<td>LISTVOLUME</td>
<td>LV</td>
<td>Display volume information</td>
</tr>
<tr>
<td></td>
<td>LISTVRS</td>
<td>LS</td>
<td>Display vital record specification</td>
</tr>
<tr>
<td>Search</td>
<td>SEARCHBIN</td>
<td>SB</td>
<td>Create a list of bin numbers</td>
</tr>
<tr>
<td></td>
<td>SEARCHDATASET</td>
<td>SD</td>
<td>Create a list of data sets</td>
</tr>
<tr>
<td></td>
<td>SEARCHOWNER</td>
<td>SO</td>
<td>Create a list of owners</td>
</tr>
<tr>
<td></td>
<td>SEARCHPRODUCT</td>
<td>SP</td>
<td>Create a list of software products</td>
</tr>
<tr>
<td></td>
<td>SEARCHRACK</td>
<td>SR</td>
<td>Create a list of rack numbers</td>
</tr>
<tr>
<td></td>
<td>SEARCHVOLUME</td>
<td>SV</td>
<td>Create a list of volumes</td>
</tr>
<tr>
<td></td>
<td>SEARCHVRS</td>
<td>SS</td>
<td>Create a list of vital record specifications</td>
</tr>
</tbody>
</table>

Refer to [z/OS DFSMSrmm Managing and Using Removable Media](https://www.ibm.com) 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.

**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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of a 72-byte standard save area in the primary address space</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

```
name
APIADDR=apiaddr

,OPERATION=BEGIN
parameters-1
,OPERATION=CONTINUE,OUTBUFADDR=outbufaddr-,TOKEN=token
,OPERATION=RELEASE-,TOKEN=token
,OPERATION=ENDALL

,RETCODE=retcode,RSNCODE=rsncode

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0

,MF=S
(1)
,MF=(L,,list addr,,attr)
,COMPLETE
(2)
,NOCHECK
,COMPLETE
(3)
,NOCHECK

parameters-1:
SUBCMDADDR=subcmdaddr,OUTBUFADDR=outbufaddr

,OUTPUT=FIELDS,EXPAND=YES,MULTI=NO,TOKEN=token
,OUTPUT=LINES
```

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 of X'0F' to force the parameter list to a word boundary or X'0D' to force the parameter list to a doubleword boundary. If you do not code attr, the system provides a value of X'0D'.

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

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

**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 32761.

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>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
<th>Related Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>—</td>
<td><strong>Meaning</strong>: Success.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Refer to the action provided with the specific reason code.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td><strong>Meaning</strong>: EDGXCI command is successfully completed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: None required.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td><strong>Meaning</strong>: There is more output waiting to be given to you.</td>
<td>EDG3900I</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: After you have processed the output in your output buffer, use OPERATION=CONTINUE to get more output.</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>—</td>
<td><strong>Meaning</strong>: Program error. An exception condition has been encountered, but the operation you requested was completed. The output results might not be acceptable to you.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Refer to the action provided with the specific reason code.</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>02</td>
<td><strong>Meaning</strong>: There is nothing to CONTINUE.</td>
<td>EDG3901I</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: None required.</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>—</td>
<td><strong>Meaning</strong>: Program error. An error condition has been encountered, and the operation you requested was not successfully completed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: Refer to the action provided with the specific reason code.</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>02</td>
<td><strong>Meaning</strong>: Required token is missing.</td>
<td>EDG3902E</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: You need to use TOKEN=token</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>04</td>
<td><strong>Meaning</strong>: Required address of the input subcommand is missing.</td>
<td>EDG3903E</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action</strong>: You need to use SUBCMDADDR=subcmdaddr</td>
<td></td>
</tr>
<tr>
<td>Return Code</td>
<td>Reason Code</td>
<td>Meaning and Action</td>
<td>Related Message</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>--------------------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>
| 108         | 06          | **Meaning:** Required address of your output buffer is missing.  
**Action:** Use OUTBUFAADDR=\texttt{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=\texttt{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=\texttt{subcmdaddr}.  
**Action:** Use OUTPUT=\texttt{LINES} or specify a different subcommand. | EDG3911E |
| 108         | 60          | **Meaning:** The length of the subcommand specified by SUBCMDADDR=\texttt{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)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
<th>Related Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>02</td>
<td>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.</td>
<td>EDG3913E</td>
</tr>
<tr>
<td>116</td>
<td>—</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>02</td>
<td>Meaning: DFSMSrmm is not installed. Action: Ensure DFSMSrmm is installed and active before running your program.</td>
<td>EDG3914E</td>
</tr>
<tr>
<td>116</td>
<td>04</td>
<td>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.</td>
<td>EDG3915E</td>
</tr>
<tr>
<td>116</td>
<td>06</td>
<td>Meaning: An abnormal end has occurred. Action: Remove the cause of the abnormal end. Rerun your program.</td>
<td>EDG3916E</td>
</tr>
<tr>
<td>120</td>
<td>02</td>
<td>Meaning: Program error has occurred while you were using the high-level API. Action: Refer to the action provided with the specific reason code.</td>
<td>EDG3918E</td>
</tr>
<tr>
<td>120</td>
<td>04</td>
<td>Meaning: The LOAD for program EDGXAPI failed. Action: Correct the cause of the error and retry the command.</td>
<td>EDG3919E</td>
</tr>
</tbody>
</table>

EDGXCI Example

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

- Obtain space for your output buffer in your work area in dynamic storage.
- Obtain space for the parameter list in your work area in dynamic storage.
- Specify subcommands that have 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)
Figure 2. Communicating with the DFSMSrmm API (Part 2 of 3)
** Call API to release resource such as storage and loaded modules. **
** ***********************************************************************

RELEASE DS OH
REL1 EDGXCI MF=(E,MYPL,NOCHECK),PLISTVER=MAX,
       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.
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.
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.
#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)
```c
int main(int argc, char* argv [])
{
    long rc = 0;
    FILE* xmlFp;
    RmmApi* pApi;
    RmmCommand* pCom;
    char* tsoCommand;
    tsoCommand = argv[1];

    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);
    }

    pApi = new RmmApi();
    printf("RmmApi object created\n");
    pCom = new RmmCommand(pApi);
}
```

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

close(sfiFp);
close(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

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RmmInterface</td>
<td>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.</td>
</tr>
<tr>
<td>RmmApi</td>
<td>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.</td>
</tr>
<tr>
<td>RmmCommand</td>
<td>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.</td>
</tr>
<tr>
<td>RmmTransaction</td>
<td>This class makes use of the RmmApi and RmmCommand classes. Instantiate an instance of this class, if you want to use the runCommandXml method.</td>
</tr>
</tbody>
</table>

Java class

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

Table 6. DFSMSrmm API Command Java Class

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RmmJApi</td>
<td>Instantiate an instance of this class to communicate with DFSMSrmm from a Java application.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RmmApi.openApi()</td>
<td>Use this method to check that DFSMSrmm is active and available to process commands.</td>
</tr>
<tr>
<td>RmmApi.closeApi()</td>
<td>Use this method when you no longer want to communicate with DFSMSrmm using this command session.</td>
</tr>
<tr>
<td>RmmCommand.issueCmd()</td>
<td>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.</td>
</tr>
<tr>
<td>RmmCommand.getBufferSfi()</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table 7. DFSMSrmm API C++ Methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RmmCommand.getBufferXml()</code></td>
<td>Use this method to obtain a string that contains the XML output converted from the SFI output of subcommand processing.</td>
</tr>
<tr>
<td><code>RmmCommand.getNextEntry()</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>RmmInterface.getMessageText()</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>RmmInterface.getApiRc()</code></td>
<td>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 <a href="#">EDGXC1</a> Return and Reason Codes” on page 10 for information about message processing.</td>
</tr>
<tr>
<td><code>RmmInterface.getApiRs()</code></td>
<td>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 <a href="#">EDGXC1</a> Return and Reason Codes” on page 10 for information about message processing.</td>
</tr>
<tr>
<td><code>RmmTransaction.runCommandXml()</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
| `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:  
  - 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RmmJApi.runCommandXml()</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 8. DFSMSrmr API Java Methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RmmJApi.runCommandXmlShort()</td>
<td>Use this method to return a string containing the XML output for key values only. Only specific search commands return key fields. For example:</td>
</tr>
<tr>
<td></td>
<td>• For SearchVolume, only the volser is returned.</td>
</tr>
<tr>
<td></td>
<td>• For SearchDataset, only the datasetname, volume, and filesequence number are returned.</td>
</tr>
<tr>
<td></td>
<td>• For SearchOwner, only the owner ID is returned.</td>
</tr>
<tr>
<td></td>
<td>• For SearchRack/SearchBin, only the rack/bin number, location, and media name are returned.</td>
</tr>
<tr>
<td></td>
<td>Other commands work as well, but they return all of the data, not just the key values.</td>
</tr>
</tbody>
</table>

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.

```c++
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.
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; &
&amp;lt; &lt;
&amp;gt; &gt;
&amp;quot; &quot;
&amp;apos; &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 EDGXC1 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 installation 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 rmmapitc.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 e/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 RmmApi.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.

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

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**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>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.</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>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 the search criteria during search processing.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Variation</th>
<th>Guidelines</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single parameter list and a single token area</td>
<td>• Only one subcommand request can be active at a time.</td>
<td>&quot;Coding a Single Parameter List, Single Token Area&quot; on page 36</td>
</tr>
<tr>
<td></td>
<td>• An active subcommand request must be completed before beginning another subcommand request.</td>
<td></td>
</tr>
<tr>
<td>Single parameter list and multiple token area</td>
<td>• More than one subcommand request can be active at a time.</td>
<td>&quot;Coding a Single Parameter List, Multiple Token Areas&quot; on page 38</td>
</tr>
<tr>
<td></td>
<td>• Only one subcommand request can be processed at any given time.</td>
<td></td>
</tr>
<tr>
<td>Multiple parameter lists with a single token area</td>
<td>• Only one subcommand can be active at a time.</td>
<td>&quot;Coding Multiple Parameter List, Single Token Area&quot; on page 40</td>
</tr>
<tr>
<td></td>
<td>• Different parameter lists can be used for these tasks:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Begin subcommand requests.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Continue subcommand requests.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Release resources.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Starting a new subcommand request ends any previous subcommand request.</td>
<td></td>
</tr>
</tbody>
</table>
Table 10. Types of Parameter Lists (continued)

<table>
<thead>
<tr>
<th>Variation</th>
<th>Guidelines</th>
<th>Reference</th>
</tr>
</thead>
</table>
| Multiple parameter lists and multiple token area | • More than one subcommand request can be active at a time.  
• More than one active subcommand request can be processed at a time.  
• Different parameter lists can be used to:  
  – Begin subcommand requests.  
  – Continue subcommand requests.  
  – Release resources.          | "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.
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)
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

EDGXI 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.

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 **
----------------------------------

** Continue the subcommand **

** Done with the subcommand, release **

** Start the second subcommand **

** Continue the subcommand **

** Done with the subcommand, release **

Figure 13. Multiple Parameter Lists, Single Token Area
Figure 14. Multiple Parameter Lists, Multiple Token Area (Part 1 of 2)
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.
Specifying TOKEN=TOKENA on the EXECUTE form of EDGXCI causes the 4-byte TOKENA area to be set to all zeros upon return from freeing the token. TOKEN=token is required even when you specify MF=(E,label,NOCHECK), unless you also specify OPERATION=ENDALL. Specifying TOKEN=token causes the 4-byte token area to be updated upon return from the DFSMSrmm API. The token is set to all zeros by the EDGXCI macro expansion.

- 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.

---

Specifying TOKEN=TOKENA on the EXECUTE form of EDGXCI causes the 4-byte TOKENA area to be set to all zeros upon return from freeing the token. Specifying 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).
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 EDGXC1 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.

```assembly
//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 **
//**
//** SYSMOD: 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.SCEECCP, 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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Set from</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function code</td>
<td>1. Open API (start communication)</td>
<td>User program</td>
</tr>
<tr>
<td></td>
<td>2. Close API (end communication)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. Issue command (begin a request)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. Get next buffer (continue a request)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. Release (end a request)</td>
<td></td>
</tr>
<tr>
<td>Pointer to the command</td>
<td>The user program needs to obtain the storage for a buffer big enough to hold</td>
<td>User program</td>
</tr>
<tr>
<td>buffer</td>
<td>the TSO subcommand to be issued. Maximum is 255 byte. EDGXHINT will read</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the TSO command from this buffer.</td>
<td></td>
</tr>
<tr>
<td>Pointer to the output</td>
<td>The user program needs to obtain the storage for an output buffer. Minimum</td>
<td>User program</td>
</tr>
<tr>
<td>buffer</td>
<td>recommended is 80KB. EDGXHINT will use this buffer to return the data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>requested.</td>
<td></td>
</tr>
<tr>
<td>Pointer to first</td>
<td>The user program must obtain the storage for a 256 byte buffer. This buffer</td>
<td>User program</td>
</tr>
<tr>
<td>message buffer</td>
<td>should always be cleared before EDGXHINT is called, to delete pre-existing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>content. EDGXHINT will use this buffer to return a message resulting from the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>last issued command, if appropriate.</td>
<td></td>
</tr>
<tr>
<td>Pointer to second</td>
<td>The user program must obtain the storage for a 256 byte buffer. This buffer</td>
<td>User program</td>
</tr>
<tr>
<td>message buffer</td>
<td>should always be cleared before EDGXHINT is called, to delete pre-existing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>content. EDGXHINT will use this buffer to return a second message resulting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>from the last issued command, if appropriate.</td>
<td></td>
</tr>
<tr>
<td>Message count</td>
<td>Number of messages returned by EDGXHINT</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td>API address</td>
<td>Address of EDGXAPI. Set by OPEN function. Can be used to determine if the</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td></td>
<td>API is open. If not NULL, then API is open.</td>
<td></td>
</tr>
<tr>
<td>MTAB address</td>
<td>Address of the DFSMSrmm message table. Set by OPEN function, used by</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td></td>
<td>EDGXHINT internally.</td>
<td></td>
</tr>
<tr>
<td>CMSG address</td>
<td>Address of the DFSMSrmm message routine. Set by OPEN function, used by</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td></td>
<td>EDGXHINT internally.</td>
<td></td>
</tr>
<tr>
<td>Token</td>
<td>Token used by macro EDGXCI to identify the request. The token is created at</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td></td>
<td>BEGIN processing (function 3) and used by CONTINUE processing (function 4).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The token is cleared (set to zero) by EDGXHINT after RELEASE processing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(function 5).</td>
<td></td>
</tr>
<tr>
<td>Return code</td>
<td>API return code</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td>Reason code</td>
<td>API reason code</td>
<td>EDGXHINT</td>
</tr>
</tbody>
</table>
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_outp).

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

typedef struct t_outph
{
    long out_length; // length of the output buffer
    long out_needed; // output buffer length needed
    long out_used; // output buffer length used
};
t_outph outputHeaderStruct; // variable of type t_outph

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

typedef struct t_interface
{
    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 EDGCMSG
    long token; // token to identify the request
    long returncode; // API return code
    long reasoncode; // API reason code
};
t_interface interStruct; // 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:

```c
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:

```c
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.

```c
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:

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFI</td>
<td>Structured Field Introducer. A structure with a minimum size of 8 bytes in this format:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Two-byte length. The length includes the length of the SFI (8 bytes) and the length of the data following the SFI.</td>
</tr>
<tr>
<td>3</td>
<td>Three-byte SFI identifier (ID)</td>
</tr>
</tbody>
</table>
One-byte SFI type modifier
One-byte (reserved)
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 EDGXC1 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 EDGXC1 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.
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.
**Figure 21. Example of Output Using OUTPUT=FIELDS**

- 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.

```
<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) >
<CTDJ - 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) >
<LRL - 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 >
<VTPY - 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 Description : 32, character >
<ABND - Closed by Abend : 1, bin(8) >
<CTLG - Catalog status : 1, bin(8) >
<ZJBN - Secondary VRS jobname : 8, character >
<2MME - Secondary VRS mask : 8, character >
<2SCN - Secondary VRS subchain name : 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 >
<DPC - Percent of volume : 1, bin(8) >
<XDTJ - Expiration Date : 4, packed decimal >
<XDTJ - Original Expiration Date : 4, packed decimal >
<LDTND - Deleted By Disposition Pr : 1, bin(8) >
<DSSG - Data Set Size : 14, compound >
<End DATASET Group>
```
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 SEARCHVRGS 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.

### 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:

```
<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
<DESEQ - Data Set Sequence : 4, bin(31) >x'00000001'
<TZ - Time Zone : 4, bin(31) >x'FFFFFF99'
<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'00000001'
<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 >
<DTLJ - 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 >
<VSRD - Primary VRS Subchain da : 4, packed decimal >
<VSCD - Secondary VRS Subchain da : 4, packed decimal >
<NME - Security Class Name : 8, character >
<CLS - Security Class Description : 32, character >
<ABND - Closed by Abend : 1, bin(8) >x'00'
<CTLG - Catalog status : 1, bin(8) >x'00'
<ZJBN - Secondary VRS Jobname mas : 8, character >
<ZMME - Secondary VRS mask : 8, character >
<ZSCN - Secondary VRS subchain na : 8, character >
<ZSCD - Secondary VRS subchain da : 4, packed decimal >
<BLKT - Total block count : 4, bin(31) >x'00000001'
<CPGM - Creating program name : 8, character >IEBGENER
<LPDM - 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
<DCTP - 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'
<DS56 - Data Set Size : 14, compound >x'010303010A060000000017D78'
<End DATASET Group>
```
• 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 98ccccc as CYCL/ccccc retention date format.
• Interpret 0000098 as CATRETPD retention date format.
• Interpret yyyyddd as yyyy/mm/dd, yyyy/dd/mm, mm/dd/yyyy, dd/mm/yyyy, dd/yyyy/mm, mm/yyyy/dd.

DFSMSrmm also returns time in packed decimal format: hhmmssstC, where hhmmssst 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 CHANGECOMPONENT, you can specify the TZ operand to indicate to the DFSMSrmm system the time zone offset the application is using. DFSMSrmrmm 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:

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

where “xxxx” is a character type of mnemonic. In your application program, you need to use the 3-byte 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.


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.
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.

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.

Groups and subgroups, such as MESSAGE and SECCLES, 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.
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.

```
<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**

<table>
<thead>
<tr>
<th>Message</th>
<th>SFI ID(s)</th>
<th>Subcommand(s)</th>
<th>RC</th>
<th>RSN(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDG3010</td>
<td>ENTN</td>
<td>All SEARCH subcommands when no (0) entry is returned</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>EDG3011</td>
<td>ENTN</td>
<td>All SEARCH subcommands when 1 entry returned</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>2 and 4</td>
</tr>
<tr>
<td>EDG3012</td>
<td>ENTN</td>
<td>All SEARCH subcommands when &gt; 1 entry returned</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>2 and 4</td>
</tr>
<tr>
<td>EDG3013</td>
<td>VOL</td>
<td>AV</td>
<td>12</td>
<td>many</td>
</tr>
<tr>
<td>EDG3014</td>
<td>CNT</td>
<td>AV</td>
<td>12</td>
<td>many</td>
</tr>
<tr>
<td>EDG3015</td>
<td>OWN VOL</td>
<td>GV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EDG3016</td>
<td>RCK</td>
<td>AV CV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EDG3017</td>
<td>RCK</td>
<td>AB AR</td>
<td>12</td>
<td>18 68 70</td>
</tr>
<tr>
<td>EDG3018</td>
<td>CNT</td>
<td>AB AR</td>
<td>12</td>
<td>18 68 70</td>
</tr>
<tr>
<td>EDG3019</td>
<td>RCK</td>
<td>DB DR</td>
<td>12</td>
<td>many</td>
</tr>
<tr>
<td>EDG3020</td>
<td>CNT</td>
<td>DB DR</td>
<td>12</td>
<td>many</td>
</tr>
<tr>
<td>EDG3025</td>
<td>CONT</td>
<td>All SEARCH subcommands</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>EDG3277</td>
<td>FRC FRS</td>
<td>AV CV</td>
<td>12</td>
<td>122</td>
</tr>
<tr>
<td>EDG3278</td>
<td>CSG</td>
<td>AV CV</td>
<td>12</td>
<td>124</td>
</tr>
<tr>
<td>EDG3288</td>
<td>FRC FRS VOL</td>
<td>CV DV</td>
<td>12</td>
<td>132</td>
</tr>
<tr>
<td>EDG3289</td>
<td>FRC FRS</td>
<td>CV</td>
<td>12</td>
<td>134</td>
</tr>
<tr>
<td>EDG3292</td>
<td>CLIB</td>
<td>AV CV</td>
<td>12</td>
<td>140</td>
</tr>
<tr>
<td>EDG3301</td>
<td>FRC FRS</td>
<td>AV CV GV</td>
<td>12</td>
<td>152</td>
</tr>
<tr>
<td>EDG3310</td>
<td>CLIB</td>
<td>CV DV</td>
<td>12</td>
<td>170</td>
</tr>
<tr>
<td>EDG3311</td>
<td>FRC FRS</td>
<td>AV CV DV</td>
<td>12</td>
<td>172</td>
</tr>
<tr>
<td>EDG3314</td>
<td>MEDN</td>
<td>CV</td>
<td>12</td>
<td>176</td>
</tr>
<tr>
<td>EDG3328</td>
<td>KEYF</td>
<td>SD 5V</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

For a detailed explanation of these messages, see [z/OS MVS System Messages, Vol 5 (EDG-GFS)](https://www.ibm.com/support/docview.ws/docview/72572). 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](https://www.ibm.com/support/docview.ws/docview/72401).

### 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.
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: ADDBIN, ADDDATASET, ADDOWNER, ADDPRODUCT, ADDRACK, ADDVOLUME, and ADDVRS. You use these subcommands to add information to the DFSMSrmm control data set.

The DFSMSrmm API returns information 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.

**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.
DELETE-Type of Subcommands

The DFSMSrmm DELETE-type of subcommands are: DELETEBIN, DELETEDATASET, DELETEOwner, DELETEPRODUCT, DELETERACK, DELETENVOLUME, 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:

1. The GETVOLUME request was successful. The DFSMSrmm API returns volume information and owner information as shown in Figure 34.
2. 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.

LIST-Type of Subcommands

The DFSMSrmm LIST-type of subcommands are: LISTBIN, LISTCONTROL, LISTDATASET, LISTOWNER, LISTPRODUCT, LISTTRACK, 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:

1. “LISTBIN SFIs”
2. “LISTCONTROL SFIs” on page 64
3. “LISTDATASET SFIs” on page 68
4. “LISTOWNER SFIs” on page 69
5. “LISTPRODUCT SFIs” on page 70
6. “LISTTRACK SFIs” on page 70
7. “LISTVOLUME SFIs” on page 71
8. “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.
LISTCONTROL SFIs

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

Figure 35. SFIs for LISTBIN with OUTPUT=FIELDS

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 36. SFIs for LISTCONTROL with OUTPUT=FIELDS (Part 1 of 4)
Figure 36. SFIs for LISTCONTROL with OUTPUT=FIELDS (Part 2 of 4)
Figure 36. SFIs for LISTCONTROL with OUTPUT=FIELDS (Part 3 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.
LISTOWNER SFIs

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

LISTTRACK SFIs
The SFIs produced for the LISTTRACK 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 LISTTRACK with OUTPUT=FIELDS
LISTVOLUME SFIs

The SFIs produced for the LISTVOLUME subcommand with OUTPUT=FIELDS are shown in Figure 41 on page 72.
Figure 41. SFIs for LISTVOLUME with OUTPUT=FIELDS (Part 1 of 2)
The SFIs produced for the LISTVRS subcommand with OUTPUT=FIELDS are shown in Figure 42 on page 74.

**LISTVRS SFIs**

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

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.
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.

<xbegin product group>
<?num - software product number : 8, character >
<?ver - software product version : 6, character >
<?own - owner : 8, character >
<?pme - product software name : 30, character >
<?pdsc - product description : 32, character >
<?vln - number of volumes : 4, bin(31)>
<?tz - time zone : 4, bin(31)>
<xend product group>

<xbegin prodvol group>
<?vol - volume serial : 6, character >
<?rck - rack or bin number : 6, character >
<?fcd - product feature code : 4, character >
<xend prodvol group>

<xend 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.

<xbegin 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)>
<xend 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.
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.

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.

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 and 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 Mediamame 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.
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.

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:

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>2-byte length: SFI length plus data length</td>
</tr>
<tr>
<td>2-4</td>
<td>3-byte identifier: SFI ID (hexadecimal)</td>
</tr>
<tr>
<td>5</td>
<td>1-byte type modifier: Type of SFI</td>
</tr>
<tr>
<td></td>
<td>• 0 = 8-byte, fixed-length SFI</td>
</tr>
<tr>
<td>6</td>
<td>1-byte (Reserved)</td>
</tr>
<tr>
<td>7</td>
<td>1-byte data type: Type of data, if any, that follows the SFI</td>
</tr>
<tr>
<td></td>
<td>• 0 = Undefined (no data)</td>
</tr>
<tr>
<td></td>
<td>• 1 = Character (fixed-length)</td>
</tr>
<tr>
<td></td>
<td>• 2 = Bit(8) (1-byte flag, multiple bits can be on)</td>
</tr>
<tr>
<td></td>
<td>• 3 = Binary(8) (1-byte (hex) value)</td>
</tr>
<tr>
<td></td>
<td>• 4 = Binary(15) (2-byte (hex) value)</td>
</tr>
<tr>
<td></td>
<td>• 5 = Binary(32) (4-byte (hex) unsigned value)</td>
</tr>
<tr>
<td></td>
<td>• 6 = Binary(64) (8-byte (hex) value)</td>
</tr>
<tr>
<td></td>
<td>• 7 = Character (variable-length)</td>
</tr>
<tr>
<td></td>
<td>• 8 = Compound SFI (multiple related values, see &quot;Compound SFI&quot; on page 84.)</td>
</tr>
<tr>
<td></td>
<td>• 9 = (4 bytes) Packed decimal Julian date: yyydddC</td>
</tr>
<tr>
<td></td>
<td>• A = (4 bytes) Packed decimal time format: hhmmsstC</td>
</tr>
</tbody>
</table>

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:

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

**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*

<table>
<thead>
<tr>
<th>Begin - End IDs</th>
<th>Resource Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'021000' - X'021080'</td>
<td>ACCESS - within VOLUME</td>
</tr>
<tr>
<td>X'022000' - X'022080'</td>
<td>ACTIONS - within CONTROL</td>
</tr>
<tr>
<td>X'024000' - X'024080'</td>
<td>CNTL - within CONTROL</td>
</tr>
<tr>
<td>X'025000' - X'025080'</td>
<td>CONTROL</td>
</tr>
<tr>
<td>X'026000' - X'026080'</td>
<td>DATASET</td>
</tr>
<tr>
<td>X'027000' - X'027080'</td>
<td>LOCDEF - within CONTROL</td>
</tr>
<tr>
<td>X'027500' - X'027580'</td>
<td>MEDINF - within CONTROL</td>
</tr>
<tr>
<td>X'028000' - X'028080'</td>
<td>MESSAGE</td>
</tr>
<tr>
<td>X'029000' - X'029080'</td>
<td>MNTMSG - within CONTROL</td>
</tr>
</tbody>
</table>
Table 13. Begin and End Group Structured Field Introducers (continued)

<table>
<thead>
<tr>
<th>Begin - End IDs</th>
<th>Resource Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'02A000' - X'02A080'</td>
<td>MOVES - within CONTROL</td>
</tr>
<tr>
<td>X'03A000' - X'03A080'</td>
<td>OPENRULE within CONTROL</td>
</tr>
<tr>
<td>X'02B000' - X'02B080'</td>
<td>OPTION - within CONTROL</td>
</tr>
<tr>
<td>X'02C000' - X'02C080'</td>
<td>OWNER</td>
</tr>
<tr>
<td>X'02D000' - X'02D080'</td>
<td>PRODUCT</td>
</tr>
<tr>
<td>X'039000' - X'039080'</td>
<td>PRODVOL - within PRODUCT</td>
</tr>
<tr>
<td>X'03B000' - X'03B080'</td>
<td>PRITITION within CONTROL</td>
</tr>
<tr>
<td>X'02E000' - X'02E080'</td>
<td>RACK or BIN</td>
</tr>
<tr>
<td>X'02F000' - X'02F080'</td>
<td>REJECT - within CONTROL</td>
</tr>
<tr>
<td>X'030000' - X'030080'</td>
<td>SECCLS - within CONTROL</td>
</tr>
<tr>
<td>X'031000' - X'031080'</td>
<td>SECLVL - within CONTROL</td>
</tr>
<tr>
<td>X'032000' - X'032080'</td>
<td>STAT - within VOLUME</td>
</tr>
<tr>
<td>X'033000' - X'033080'</td>
<td>STORE - within VOLUME</td>
</tr>
<tr>
<td>X'034000' - X'034080'</td>
<td>SYSRETC</td>
</tr>
<tr>
<td>X'035000' - X'035080'</td>
<td>VLPOOL - within CONTROL</td>
</tr>
<tr>
<td>X'036000' - X'036080'</td>
<td>VOL - within VOLUME</td>
</tr>
<tr>
<td>X'037000' - X'037080'</td>
<td>VOLUME</td>
</tr>
<tr>
<td>X'038000' - X'038080'</td>
<td>VRS</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SFI Number</th>
<th>SFI Name</th>
<th>SFI Length</th>
<th>SFI Data Type</th>
<th>Data Description</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'400000'</td>
<td>FRC</td>
<td>12</td>
<td>Binary(32)</td>
<td>Function return code</td>
<td>AV CV DV GV</td>
</tr>
<tr>
<td>X'401000'</td>
<td>FRS</td>
<td>12</td>
<td>Binary(32)</td>
<td>Function reason code</td>
<td>AV CV DV GV</td>
</tr>
<tr>
<td>X'402000'</td>
<td>RSNC</td>
<td>12</td>
<td>Binary(32)</td>
<td>Reason code</td>
<td>Any subcommand</td>
</tr>
<tr>
<td>X'403000'</td>
<td>RTNC</td>
<td>12</td>
<td>Binary(32)</td>
<td>Return code</td>
<td>Any subcommand</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>SFI Number</th>
<th>SFI Name</th>
<th>SFI Length</th>
<th>SFI Data Type</th>
<th>Data Description</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'051000'</td>
<td>MSGL</td>
<td>259</td>
<td>Character (variable length)</td>
<td>Message line</td>
<td>Any subcommand</td>
</tr>
<tr>
<td>X'052000'</td>
<td>MSGN</td>
<td>16</td>
<td>Character (fixed length)</td>
<td>Message number ID</td>
<td>As previously defined</td>
</tr>
<tr>
<td>X'053000'</td>
<td>ENTN</td>
<td>12</td>
<td>Binary(32)</td>
<td>Number of entries</td>
<td>As previously defined</td>
</tr>
<tr>
<td>X'054000'</td>
<td>KEYF</td>
<td>65</td>
<td>Character (variable length)</td>
<td>Key from</td>
<td>SD SV</td>
</tr>
<tr>
<td>X'054200'</td>
<td>KEYT</td>
<td>65</td>
<td>Character (variable length)</td>
<td>Key to</td>
<td>SD SV</td>
</tr>
<tr>
<td>X'055000'</td>
<td>TYPF</td>
<td>16</td>
<td>Character (variable length)</td>
<td>VOLUME or DATASET</td>
<td>SD SV</td>
</tr>
<tr>
<td>X'055200'</td>
<td>TYPT</td>
<td>16</td>
<td>Character (variable length)</td>
<td>VOLUME or DATASET</td>
<td>SD SV</td>
</tr>
<tr>
<td>X'057000'</td>
<td>CONT</td>
<td>92</td>
<td>Character (variable length)</td>
<td>SEARCH Continue information</td>
<td>All search subcommands</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SFI Number</th>
<th>SFI Name</th>
<th>SFI Length</th>
<th>SFI Data Type</th>
<th>Data Description</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>X’800500’</td>
<td>ABND</td>
<td>9</td>
<td>Binary(8)</td>
<td>Closed by Abend 0=NO 1=YES</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X’800800’</td>
<td>ACCT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Accounting source 0=JOB 1=STEP</td>
<td>LC</td>
</tr>
<tr>
<td>X’801000’</td>
<td>ACN</td>
<td>48</td>
<td>Character (variable length)</td>
<td>Account number</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X’801800’</td>
<td>ACS</td>
<td>9</td>
<td>Binary(8)</td>
<td>SMSACS 0=NO 1=YES</td>
<td>LC</td>
</tr>
<tr>
<td>X’802000’</td>
<td>ACT</td>
<td>9</td>
<td>Bit(8)</td>
<td>Actions on release '80'=SCRATCH '40'=REPLACE '20'=INIT '10'=ERASE '08'=RETURN '04'=NOTIFY For LC VLPOOL X’00’, X’04’</td>
<td>LC LV SV(e)</td>
</tr>
<tr>
<td>X’803001’</td>
<td>ADL</td>
<td>48</td>
<td>Character (variable length)</td>
<td>Address line. The SFI is incremented by one for each ADL line that is found. (X’803001’ - X’803003’)</td>
<td>LO</td>
</tr>
<tr>
<td>X’804000’</td>
<td>ADTJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Assigned date</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X’805000’</td>
<td>AST</td>
<td>9</td>
<td>Bit(8)</td>
<td>Action status '80'=PENDING '40'=CONFIRMED '20'=COMPLETE '10'=UNKNOWN</td>
<td>LC</td>
</tr>
<tr>
<td>X’806000’</td>
<td>ATM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Assigned time</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X’807000’</td>
<td>AUD</td>
<td>10</td>
<td>Binary(15)</td>
<td>SMF audit record type: 128-255, 42, or 0</td>
<td>LC</td>
</tr>
<tr>
<td>X’808000’</td>
<td>AVL</td>
<td>9</td>
<td>Bit(8)</td>
<td>Volume availability '40'=PENDING_RELEASE '20'=VITAL_RECORD '08'=ON_LOAN '04'=OPEN</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X’809000’</td>
<td>BDTJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Last control data set backup date</td>
<td>LC</td>
</tr>
<tr>
<td>X’80A000’</td>
<td>BIN</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>6-character alphanumeric bin number</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X’80B000’</td>
<td>BKPP</td>
<td>16</td>
<td>Character (Variable length)</td>
<td>Backup procedure name</td>
<td>LC</td>
</tr>
<tr>
<td>SFI Number</td>
<td>SFI Name</td>
<td>SFI Length</td>
<td>SFI Data Type</td>
<td>Data Description</td>
<td>Subcommand</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>------------</td>
<td>---------------</td>
<td>-----------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>X'80C000'</td>
<td>BLKC</td>
<td>12</td>
<td>Binary(32)</td>
<td>Block count</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'80D000'</td>
<td>BLKS</td>
<td>12</td>
<td>Binary(32)</td>
<td>Block size</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'80D030'</td>
<td>BLKT</td>
<td>12</td>
<td>Binary(32)</td>
<td>Total block count</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'80E000'</td>
<td>BLP</td>
<td>9</td>
<td>Binary(8)</td>
<td>BLP option:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=RMM 1=NORMM</td>
<td></td>
</tr>
<tr>
<td>X'80F000'</td>
<td>BMN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Bin number media name</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'810000'</td>
<td>BTM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last control data set backup time</td>
<td>LC</td>
</tr>
<tr>
<td>X'811000'</td>
<td>CACT</td>
<td>9</td>
<td>Bit(8)</td>
<td>Control active functions</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'80'=BACKUP '40'=RESTORE '20'=VERIFY</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'08'=EXTRACT '04'=DSTORE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'02'=VRSEL</td>
<td></td>
</tr>
<tr>
<td>X'811800'</td>
<td>CATS</td>
<td>9</td>
<td>Binary(8)</td>
<td>CATSYSID value</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=SET 1=NOTSET 2=*</td>
<td></td>
</tr>
<tr>
<td>X'812000'</td>
<td>CDS</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Control data set identifier</td>
<td>LC</td>
</tr>
<tr>
<td>X'812900'</td>
<td>CDSQ</td>
<td>9</td>
<td>Binary(8)</td>
<td>Control data set ENQ</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=Disabled 1=Enabled</td>
<td></td>
</tr>
<tr>
<td>X'812A00'</td>
<td>CDSU</td>
<td>10</td>
<td>Binary(15)</td>
<td>Control data set percentage used</td>
<td>LC</td>
</tr>
<tr>
<td>X'813000'</td>
<td>CDTJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Create date</td>
<td>LD LV SD SV(e)</td>
</tr>
<tr>
<td>X'814000'</td>
<td>CJBN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Job name</td>
<td>LD LV SD(e) SV(e)</td>
</tr>
<tr>
<td>X'815000'</td>
<td>CLIB</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Current library name</td>
<td>AV CV DV</td>
</tr>
<tr>
<td>X'816000'</td>
<td>CLS</td>
<td>40</td>
<td>Character (variable length)</td>
<td>Security class description</td>
<td>LC LD LV SD(e) SV(e)</td>
</tr>
<tr>
<td>X'816900'</td>
<td>CMDD</td>
<td>9</td>
<td>Binary(8)</td>
<td>Command Authorization based on DSN:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=No 1=Yes</td>
<td></td>
</tr>
<tr>
<td>X'8169A0'</td>
<td>CMDO</td>
<td>9</td>
<td>Binary(8)</td>
<td>Command Authorization based on owner:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=No 1=Yes</td>
<td></td>
</tr>
<tr>
<td>X'817000'</td>
<td>CNT</td>
<td>12</td>
<td>Binary(32)</td>
<td>Bin, rack, or volume count:</td>
<td>AB AR AV DB DR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 5-digit</td>
<td></td>
</tr>
<tr>
<td>X'817820'</td>
<td>CPGM</td>
<td>16</td>
<td>Character (fixed length)</td>
<td>Creating program name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'818000'</td>
<td>CRP</td>
<td>12</td>
<td>Binary(32)</td>
<td>CATRETPD retention</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>period:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 4-digit</td>
<td></td>
</tr>
<tr>
<td>X'818800'</td>
<td>CSDT</td>
<td>12</td>
<td>Packed decimal Julian date</td>
<td>Catalog synchronize date</td>
<td>LC</td>
</tr>
<tr>
<td>X'819000'</td>
<td>CSG</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Current storage group name</td>
<td>AV CV</td>
</tr>
<tr>
<td>SFI Number</td>
<td>SFI Name</td>
<td>SFI Length</td>
<td>SFI Data Type</td>
<td>Data Description</td>
<td>Subcommand</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>---------------</td>
<td>------------------</td>
<td>------------</td>
</tr>
<tr>
<td>X’819200’</td>
<td>CSHN</td>
<td>71</td>
<td>Character (variable length)</td>
<td>Client/server host name 1-to-63 alphanumeric characters including hyphen, period, and blank</td>
<td>LC</td>
</tr>
<tr>
<td>X’819250’</td>
<td>CSIP</td>
<td>23</td>
<td>Character (variable length)</td>
<td>Client IP address 1-to-15 numeric characters including period and blank</td>
<td>LC</td>
</tr>
<tr>
<td>X’819400’</td>
<td>CSTM</td>
<td>12</td>
<td>Packed decimal time date</td>
<td>Catalog synchronize time</td>
<td>LC</td>
</tr>
<tr>
<td>X’819600’</td>
<td>CSVE</td>
<td>9</td>
<td>Binary(8)</td>
<td>Stacked volume enable status:  0=None 1=Enabled 2=Disabled 3=Mixed</td>
<td>LC</td>
</tr>
<tr>
<td>X’819800’</td>
<td>CTLG</td>
<td>9</td>
<td>Binary(8)</td>
<td>Catalog status:  0=UNKNOWN 1=NO 2=YES</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X’81A000’</td>
<td>CTM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Create time</td>
<td>LD LV SD SV(e)</td>
</tr>
<tr>
<td>X’81A300’</td>
<td>CTNR</td>
<td>24</td>
<td>Character (variable length)</td>
<td>In container</td>
<td>LV STORE</td>
</tr>
<tr>
<td>X’81A600’</td>
<td>DBIN</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>Numeric: 0–999999 or 6 alphanumeric character destination bin number</td>
<td>LV</td>
</tr>
<tr>
<td>X’81A700’</td>
<td>DBMN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Destination bin media name</td>
<td>LV</td>
</tr>
<tr>
<td>X’81B000’</td>
<td>DBN</td>
<td>12</td>
<td>Binary(32)</td>
<td>Bin numbers in DISTANT location: Min 0, Max 6-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X’81C000’</td>
<td>DC</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Data class name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X’81D000’</td>
<td>DD</td>
<td>16</td>
<td>Character (variable length)</td>
<td>DD name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X’81E000’</td>
<td>DDTJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Delete date or last store update date</td>
<td>LC LS SS</td>
</tr>
<tr>
<td>X’81F000’</td>
<td>DEN</td>
<td>9</td>
<td>Binary(8)</td>
<td>Media density:  0=UNDEFINED 1=1600 2=6250 3=3480 4=COMPACT</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X’820000’</td>
<td>DESC</td>
<td>38</td>
<td>Character (variable length)</td>
<td>Volume or VRS description</td>
<td>LS LV SS(e) SV(e)</td>
</tr>
<tr>
<td>X’821000’</td>
<td>DEST</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Destination name</td>
<td>LV SV</td>
</tr>
<tr>
<td>X’822000’</td>
<td>DEV</td>
<td>12</td>
<td>Character (fixed length)</td>
<td>Device number</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X’823000’</td>
<td>DLR/ DLRJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Date last referenced/read</td>
<td>LD LV SD(e) LS SS(e) SV(e)</td>
</tr>
<tr>
<td>X’823700’</td>
<td>DLTD</td>
<td>9</td>
<td>Binary(8)</td>
<td>Deleted by disposition processing:  0=NO 1=YES</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>SFI Number</td>
<td>SFI Name</td>
<td>SFI Length</td>
<td>SFI Data Type</td>
<td>Data Description</td>
<td>Subcommand</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>---------------</td>
<td>------------------</td>
<td>------------</td>
</tr>
<tr>
<td>X'824000'</td>
<td>DLWJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Date last written</td>
<td>LD LV SD(e) SV(e)</td>
</tr>
<tr>
<td>X'825000'</td>
<td>DNM</td>
<td>52</td>
<td>Character (variable length)</td>
<td>Data set name mask</td>
<td>LC</td>
</tr>
<tr>
<td>X'825E00'</td>
<td>DPCT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Percent of volume</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'826000'</td>
<td>DPT</td>
<td>48</td>
<td>Character (variable length)</td>
<td>Owner’s department</td>
<td>LO</td>
</tr>
<tr>
<td>X'827000'</td>
<td>DRP</td>
<td>12</td>
<td>Binary(32)</td>
<td>Default retention period: Min 0, Max 4-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'828000'</td>
<td>DSC</td>
<td>12</td>
<td>Binary(32)</td>
<td>Data set count: Min 0, Max 4-digit</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'829000'</td>
<td>DSEQ</td>
<td>12</td>
<td>Binary(32)</td>
<td>Data set sequence: Min 0, Max 4-digit</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'82A000'</td>
<td>DSN</td>
<td>52</td>
<td>Character (variable length)</td>
<td>Data set name</td>
<td>LD LV SD SV(e)</td>
</tr>
<tr>
<td>X'82A500'</td>
<td>DSPD</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Disposition DD name</td>
<td>LC</td>
</tr>
<tr>
<td>X'82AA00'</td>
<td>DSPM</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Disposition message prefix</td>
<td>LC</td>
</tr>
<tr>
<td>X'82B000'</td>
<td>DSR</td>
<td>9</td>
<td>Binary(8)</td>
<td>Data set recording: 0=NO 1=YES</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'82B030'</td>
<td>DSS6</td>
<td>22</td>
<td>Compound (Binary(8) Factor, Binary(64) Value)</td>
<td>Data set size, Factor: 0=bytes 1=KB 2=MB 3=GB 4=TB Value: Minimum value = 0.</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'82B200'</td>
<td>DSTT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Destination type 0=SHELF 1=STORE_BUILTIN 2=MANUAL 3=AUTO 4=STORE_BINS 5=STORE_NOBINS</td>
<td>LV</td>
</tr>
<tr>
<td>X'82BB00'</td>
<td>DSYS</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Creating system ID</td>
<td>LV, SV(e)</td>
</tr>
<tr>
<td>X'82C000'</td>
<td>DTE</td>
<td>9</td>
<td>Binary(8)</td>
<td>Installation date format: 1=A 2=E 3=I 4=J</td>
<td>LC</td>
</tr>
<tr>
<td>X'82D000'</td>
<td>DTM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last store update run time</td>
<td>LC</td>
</tr>
<tr>
<td>X'82D500'</td>
<td>EBIN</td>
<td>9</td>
<td>Binary(8)</td>
<td>Extended bin enable status 0=DISABLED 1=ENABLED</td>
<td>LC</td>
</tr>
<tr>
<td>X'82DF00'</td>
<td>EML</td>
<td>71</td>
<td>Character (variable length)</td>
<td>Owner’s e-mail address, 1 to 63 characters</td>
<td>LO SO</td>
</tr>
<tr>
<td>X'82E000'</td>
<td>EMN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Owner’s node</td>
<td>LO</td>
</tr>
</tbody>
</table>
Table 16. Command SFIs (continued)

<table>
<thead>
<tr>
<th>SFI Number</th>
<th>SFI Name</th>
<th>SFI Length</th>
<th>SFI Data Type</th>
<th>Data Description</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'82F000'</td>
<td>EMU</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Owner's user ID</td>
<td>LO</td>
</tr>
<tr>
<td>X'830000'</td>
<td>ETL</td>
<td>28</td>
<td>Character (variable length)</td>
<td>Telephone number</td>
<td>LO</td>
</tr>
<tr>
<td>X'831000'</td>
<td>FCD</td>
<td>12</td>
<td>Character (variable length)</td>
<td>Feature code</td>
<td>LP LV SP SV(e)</td>
</tr>
<tr>
<td>X'831800'</td>
<td>FCSP</td>
<td>9</td>
<td>Binary(8)</td>
<td>Catalog synchronize in progress: 0=NO 1=YES</td>
<td>LC</td>
</tr>
<tr>
<td>X'832000'</td>
<td>FDB</td>
<td>12</td>
<td>Binary(32)</td>
<td>Free bins in DISTANT location</td>
<td>LC</td>
</tr>
<tr>
<td>X'833000'</td>
<td>FILE</td>
<td>12</td>
<td>Binary(32)</td>
<td>Physical file sequence</td>
<td>LD SD</td>
</tr>
<tr>
<td>X'834000'</td>
<td>FLB</td>
<td>12</td>
<td>Binary(32)</td>
<td>Free bin numbers in LOCAL location: Min 0, Max 6-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'835000'</td>
<td>FOR</td>
<td>28</td>
<td>Character (variable length)</td>
<td>Owner's forename</td>
<td>LO</td>
</tr>
<tr>
<td>X'836000'</td>
<td>FRB</td>
<td>12</td>
<td>Binary(32)</td>
<td>Free bin numbers in REMOTE location: Min 0, Max 6-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'837000'</td>
<td>FRK</td>
<td>12</td>
<td>Binary(32)</td>
<td>Free rack numbers in library: Min 0, Max 10-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'837800'</td>
<td>GDGC</td>
<td>9</td>
<td>Binary(8)</td>
<td>GDG CYCLEBY: 0=Generation 1=Create order</td>
<td>LC</td>
</tr>
<tr>
<td>X'837805'</td>
<td>GDGD</td>
<td>9</td>
<td>Binary(8)</td>
<td>GDG DUPLICATE: 0=Bump from sub chain 1=Drop from chain 2=Keep 3=Count</td>
<td>LC</td>
</tr>
<tr>
<td>X'838000'</td>
<td>GRK</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>Generic rack number = reject prefix</td>
<td>LC</td>
</tr>
<tr>
<td>X'839000'</td>
<td>HLOC</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Home location</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'839200'</td>
<td>HLOT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Home location type 0=SHELF 1=STORE_BUILTIN 2=MANUAL 3=AUTO 4=STORE_BINS 5=STORE_NOBINS</td>
<td>LV</td>
</tr>
<tr>
<td>X'83A000'</td>
<td>INTR</td>
<td>9</td>
<td>Binary(8)</td>
<td>Volume intransit status: 0=NO 1=YES</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'83B000'</td>
<td>IPL</td>
<td>9</td>
<td>Binary(8)</td>
<td>Date check required on IPL: 0=NO 1=YES</td>
<td>LC</td>
</tr>
<tr>
<td>X'83B830'</td>
<td>IRRM</td>
<td>9</td>
<td>Binary(8)</td>
<td>Managed by IRMM), 0=NO 1=YES</td>
<td>LV, SV (e)</td>
</tr>
<tr>
<td>X'83C000'</td>
<td>ITL</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Telephone number</td>
<td>LO</td>
</tr>
<tr>
<td>X'83CA00'</td>
<td>JBDT</td>
<td>12</td>
<td>Packed decimal Julian date</td>
<td>Last Journal Backup Date</td>
<td>LC</td>
</tr>
<tr>
<td>X'83CB00'</td>
<td>JBTM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last Journal Backup Time</td>
<td>LC</td>
</tr>
<tr>
<td>SFI Number</td>
<td>SFI Name</td>
<td>SFI Length</td>
<td>SFI Data Type</td>
<td>Data Description</td>
<td>Subcommand</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>-------------------</td>
<td>-------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>X'83D000'</td>
<td>JDS</td>
<td>52</td>
<td>Character (variable length)</td>
<td>Journal name</td>
<td>LC</td>
</tr>
<tr>
<td>X'83E000'</td>
<td>JRNF</td>
<td>10</td>
<td>Binary(15)</td>
<td>JOURNALFULL parmlib value: 0-99</td>
<td>LC</td>
</tr>
<tr>
<td>X'83EA00'</td>
<td>JRNS</td>
<td>9</td>
<td>Binary(8)</td>
<td>Journal status: 0=Disabled 1=Enabled 2=Locked</td>
<td>LC</td>
</tr>
<tr>
<td>X'83ED00'</td>
<td>JRNT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Journal transaction: 0=No 1=Yes</td>
<td>LC</td>
</tr>
<tr>
<td>X'83F000'</td>
<td>JRNU</td>
<td>10</td>
<td>Binary(15)</td>
<td>Journal percentage used: 0 - 100</td>
<td>LC</td>
</tr>
<tr>
<td>X'83F500'</td>
<td>KEL1</td>
<td>72</td>
<td>Character (variable length)</td>
<td>Key encryption key label 1</td>
<td>LV, SV(e)</td>
</tr>
<tr>
<td>X'83F505'</td>
<td>KEL2</td>
<td>72</td>
<td>Character (variable length)</td>
<td>Key encryption key label 2</td>
<td>LV, SV(e)</td>
</tr>
<tr>
<td>X'83F520'</td>
<td>KEM1</td>
<td>13</td>
<td>Character (variable length)</td>
<td>Key encoding mechanism for key label 1: LABEL or HASH</td>
<td>LV, SV(e)</td>
</tr>
<tr>
<td>X'83F525'</td>
<td>KEM2</td>
<td>13</td>
<td>Character (variable length)</td>
<td>Key encoding mechanism for key label 2: LABEL or HASH</td>
<td>LV, SV(e)</td>
</tr>
<tr>
<td>X'840000'</td>
<td>LBL</td>
<td>9</td>
<td>Bit(8)</td>
<td>Volume label type: '20'=NL '10'=AL '08'=SL '02'=BLP '01'=UL</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'841000'</td>
<td>LBN</td>
<td>12</td>
<td>Binary(32)</td>
<td>Bin numbers in LOCAL location Min 0, Max 6-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'842000'</td>
<td>LCID</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Last change user ID</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'843000'</td>
<td>LCT</td>
<td>10</td>
<td>Binary(15)</td>
<td>Default lines per page Min 10, Max 3-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'843100'</td>
<td>LCTK</td>
<td>12</td>
<td>Binary (31)</td>
<td>Local tasks binary value</td>
<td>LC</td>
</tr>
<tr>
<td>X'844000'</td>
<td>LDDF</td>
<td>9</td>
<td>Binary(8)</td>
<td>Location definition exists: 0=NO 1=YES</td>
<td>LC</td>
</tr>
<tr>
<td>X'843B00'</td>
<td>LDD</td>
<td>16</td>
<td>Character (fixed length)</td>
<td>Last used DD name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'845000'</td>
<td>LDEV</td>
<td>12</td>
<td>Character (fixed length)</td>
<td>Last drive</td>
<td>LD SD(e) LV SV(e)</td>
</tr>
<tr>
<td>X'846000'</td>
<td>LDLC</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Location name</td>
<td>LC</td>
</tr>
<tr>
<td>X'847000'</td>
<td>LDLT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Location type: 0= SHELF 1= AUTO 2= MANUAL 3= STORE</td>
<td>LC</td>
</tr>
<tr>
<td>X'848000'</td>
<td>LDMN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Location media name</td>
<td>LC</td>
</tr>
<tr>
<td>SFI Number</td>
<td>SFI Name</td>
<td>SFI Length</td>
<td>SFI Data Type</td>
<td>Data Description</td>
<td>Subcommand</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>X'849000'</td>
<td>LDMT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Location management type: 0=UNDEFINED 1=BIN 2=NOBINS</td>
<td>LC</td>
</tr>
<tr>
<td>X'84A000'</td>
<td>LDPR</td>
<td>12</td>
<td>Binary(32)</td>
<td>Location priority: Min 0, Max 4-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'84B000'</td>
<td>LINE</td>
<td>264</td>
<td>Character (variable length)</td>
<td>Output data line</td>
<td>All list and search subcommands</td>
</tr>
<tr>
<td>X'84B420'</td>
<td>LJOB</td>
<td>16</td>
<td>Character (fixed length)</td>
<td>Last used job name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'84C000'</td>
<td>LOAN</td>
<td>16</td>
<td>Character (fixed length)</td>
<td>Loan location</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'84D000'</td>
<td>LOC</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Location</td>
<td>LB LR LS LV SB SR SS SV</td>
</tr>
<tr>
<td>X'84E000'</td>
<td>LOCT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Location type 0=SHELF 1=STORE_BUILTIN 2=MANUAL 3=AUTO 4=STORE_BINS 5=STORE_NOBINS 6=IN_CONTAINER</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'84E760'</td>
<td>LPGM</td>
<td>16</td>
<td>Character (fixed length)</td>
<td>Last used program name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'84F000'</td>
<td>LRCL</td>
<td>12</td>
<td>Binary(32)</td>
<td>Logical record length: Min 0, Max 5-digit</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'850000'</td>
<td>LRK</td>
<td>12</td>
<td>Binary(32)</td>
<td>Library rack numbers: Min 0, Max 10-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'850370'</td>
<td>LSTP</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Last used step name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'850500'</td>
<td>LVC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Current label version: 0=No version specified 1=Label version 1 3=Label version 3 4=Label version 4</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'850A00'</td>
<td>LVN</td>
<td>9</td>
<td>Binary(8)</td>
<td>Required label version: 0=No version specified 1=Label version 3 4=Label version 4</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'851000'</td>
<td>MC</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Management class</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'851400'</td>
<td>MDNF</td>
<td>16</td>
<td>Character (8)</td>
<td>Media Information Name</td>
<td>LV SV(e) LC</td>
</tr>
<tr>
<td>X'851980'</td>
<td>MDRA</td>
<td>12</td>
<td>Binary(32)</td>
<td>MEDINF replace policy for age</td>
<td>LC</td>
</tr>
<tr>
<td>X'8519C0'</td>
<td>MDRP</td>
<td>12</td>
<td>Binary(32)</td>
<td>MEDINF replace policy for permanent errors</td>
<td>LC</td>
</tr>
<tr>
<td>X'8519E0'</td>
<td>MDRT</td>
<td>12</td>
<td>Binary(32)</td>
<td>MEDINF replace policy for temporary errors</td>
<td>LC</td>
</tr>
<tr>
<td>X'8519F0'</td>
<td>MDRW</td>
<td>12</td>
<td>Binary(32)</td>
<td>MEDINF replace policy for write mount count</td>
<td>LC</td>
</tr>
<tr>
<td>X'851A00'</td>
<td>MDRX</td>
<td>16</td>
<td>Character (8)</td>
<td>External Recording Technology</td>
<td>LV SV(e) LC</td>
</tr>
</tbody>
</table>

Appendix A. Structured Field Introducers  93
<table>
<thead>
<tr>
<th>SFI Number</th>
<th>SFI Name</th>
<th>SFI Length</th>
<th>SFI Data Type</th>
<th>Data Description</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'852000'</td>
<td>MDS</td>
<td>52</td>
<td>Character (variable length)</td>
<td>Control data set name</td>
<td>LC</td>
</tr>
<tr>
<td>X'853000'</td>
<td>MDTJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Control data set create date</td>
<td>LC</td>
</tr>
<tr>
<td>X'853400'</td>
<td>MDTX</td>
<td>16</td>
<td>Character (8)</td>
<td>External Media Type</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'854000'</td>
<td>MEDA</td>
<td>9</td>
<td>Binary(8)</td>
<td>Media special attributes:</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'855000'</td>
<td>MEDC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Media compaction</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'856000'</td>
<td>MEDN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Media name</td>
<td>CV LC LB LR LV SB SR SV</td>
</tr>
<tr>
<td>X'857000'</td>
<td>MEDR</td>
<td>9</td>
<td>Binary(8)</td>
<td>Recording technology:</td>
<td>LV SV LC</td>
</tr>
<tr>
<td>X'858000'</td>
<td>MEDT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Media type:</td>
<td>LV SV LC</td>
</tr>
<tr>
<td>X'859000'</td>
<td>MFR</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Source location name</td>
<td>LC</td>
</tr>
<tr>
<td>X'85A000'</td>
<td>MID</td>
<td>20</td>
<td>Character (variable length)</td>
<td>Mount message ID</td>
<td>LC</td>
</tr>
<tr>
<td>X'85A500'</td>
<td>MIV</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>Moving-in volume</td>
<td>LB SB</td>
</tr>
<tr>
<td>X'85A900'</td>
<td>MOV</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>Moving-out volume</td>
<td>LB SB</td>
</tr>
<tr>
<td>X'85B000'</td>
<td>MOVM</td>
<td>9</td>
<td>Binary(8)</td>
<td>Move mode:</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'85C000'</td>
<td>MOP</td>
<td>9</td>
<td>Binary(8)</td>
<td>Master overwrite:</td>
<td>LC</td>
</tr>
<tr>
<td>X'85D000'</td>
<td>MRP</td>
<td>12</td>
<td>Binary(32)</td>
<td>Maximum retention period:</td>
<td>LC</td>
</tr>
<tr>
<td>SFI Number</td>
<td>SFI Name</td>
<td>SFI Length</td>
<td>SFI Data Type</td>
<td>Data Description</td>
<td>Subcommand</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>X'85F000'</td>
<td>MST</td>
<td>9</td>
<td>Binary(8)</td>
<td>Move status: 0=UNKNOWN 1=PENDING 2=CONFIRMED 3=COMPLETE</td>
<td>LC</td>
</tr>
<tr>
<td>X'860000'</td>
<td>MTM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Control data set create time</td>
<td>LC</td>
</tr>
<tr>
<td>X'861000'</td>
<td>MTO</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Target location name, installation defined name, SHELF, or SMS library name</td>
<td>LC</td>
</tr>
<tr>
<td>X'862000'</td>
<td>MTP</td>
<td>9</td>
<td>Binary(8)</td>
<td>Control data set type: 0=MASTER</td>
<td>LC</td>
</tr>
<tr>
<td>X'862800'</td>
<td>MTY</td>
<td>9</td>
<td>Binary(8)</td>
<td>Move type: 0=NOTRTS 1=RTS</td>
<td>LC</td>
</tr>
<tr>
<td>X'862B00'</td>
<td>MVBY</td>
<td>9</td>
<td>Binary(8)</td>
<td>Move by: 0=VOLUME 1=SET</td>
<td>LC</td>
</tr>
<tr>
<td>X'863000'</td>
<td>MVS</td>
<td>9</td>
<td>Binary(8)</td>
<td>MVS use 0=NO 1=YES</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'865000'</td>
<td>NLOC</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Required location</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'865200'</td>
<td>NLOT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Required location type 0=SHELF 1=STORE_BUILTIN 2=MANUAL 3=AUTO 4=STORE_BINS 5=STORE_NOBINS</td>
<td>LV</td>
</tr>
<tr>
<td>X'866000'</td>
<td>NME</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Security class name</td>
<td>LC LD LV SD(e) SV(e)</td>
</tr>
<tr>
<td>X'866800'</td>
<td>NOT</td>
<td>9</td>
<td>Binary(8)</td>
<td>User notification: 0=NO 1=YES</td>
<td>LC</td>
</tr>
<tr>
<td>X'867000'</td>
<td>NVL</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>Next volume serial</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'868000'</td>
<td>NVRS</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Next VRS name</td>
<td>LS SS</td>
</tr>
<tr>
<td>X'869000'</td>
<td>OAC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Owner access 0=READ 1=UPDATE 2=ALTER</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'86A000'</td>
<td>OBMN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Old bin number media name</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'86B000'</td>
<td>OBN</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>Old bin number</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'86B800'</td>
<td>OCE</td>
<td>9</td>
<td>Binary(8)</td>
<td>Volume information recorded at O/C/EOV 0=NO 1=YES</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'86C000'</td>
<td>OLOC</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Old location</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>SFI Number</td>
<td>SFI Name</td>
<td>SFI Length</td>
<td>SFI Data Type</td>
<td>Data Description</td>
<td>Subcommand</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>X'86C200'</td>
<td>OLOT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Old location type</td>
<td>LV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0= SHELF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=STORE_BUILTIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2=MANUAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3=AUTO</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4=STORE_BINS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5=STORE_NOBINS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6=IN_CONTAINER</td>
<td></td>
</tr>
<tr>
<td>X'86D000'</td>
<td>OPL</td>
<td>10</td>
<td>Binary(15)</td>
<td>Position of rack number or pool ID</td>
<td>LC Position in the message.</td>
</tr>
<tr>
<td>X'86E000'</td>
<td>OPM</td>
<td>9</td>
<td>Binary(8)</td>
<td>Operating mode</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 1, Max 3-digit</td>
<td></td>
</tr>
<tr>
<td>X'86E8A0'</td>
<td>ORIA</td>
<td>9</td>
<td>Binary(8)</td>
<td>Input action:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0= ACCEPT 1= IGNORE 2= REJECT</td>
<td></td>
</tr>
<tr>
<td>X'86E8A8'</td>
<td>ORII</td>
<td>9</td>
<td>Bit(8)</td>
<td>Input IGNORE condition (BY):</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'80'= SPECIFIC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'40'= NONSPECIFIC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'C0'= ANY</td>
<td></td>
</tr>
<tr>
<td>X'86E8B0'</td>
<td>ORIR</td>
<td>9</td>
<td>Bit(8)</td>
<td>Input REJECT condition (BY):</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'80'= SYSID X'40'= CATLG</td>
<td></td>
</tr>
<tr>
<td>X'86EA00'</td>
<td>OROA</td>
<td>9</td>
<td>Binary(8)</td>
<td>Output action:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0= ACCEPT 1= IGNORE 2= REJECT</td>
<td></td>
</tr>
<tr>
<td>X'86EA08'</td>
<td>OROI</td>
<td>9</td>
<td>Bit(8)</td>
<td>Output IGNORE condition (BY):</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'80'= SPECIFIC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'40'= NONSPECIFIC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'C0'= ANY</td>
<td></td>
</tr>
<tr>
<td>X'86EA18'</td>
<td>OROR</td>
<td>9</td>
<td>Bit(8)</td>
<td>Output REJECT condition (BY):</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'80'= SYSID X'40'= CATLG</td>
<td></td>
</tr>
<tr>
<td>X'86EF08'</td>
<td>ORTP</td>
<td>9</td>
<td>Binary(8)</td>
<td>Type of open rule entry:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=RMM 1=NORMMM</td>
<td></td>
</tr>
<tr>
<td>X'86EF80'</td>
<td>ORVS</td>
<td>14</td>
<td>Character (variable length)</td>
<td>LC</td>
<td></td>
</tr>
<tr>
<td>X'86EF85'</td>
<td>ORVL</td>
<td>14</td>
<td>Character (variable length)</td>
<td>LC</td>
<td></td>
</tr>
<tr>
<td>X'86EF8F'</td>
<td>ORVE</td>
<td>14</td>
<td>Character (variable length)</td>
<td>LC</td>
<td></td>
</tr>
<tr>
<td>X'86F000'</td>
<td>OVL</td>
<td>10</td>
<td>Binary(15)</td>
<td>Position of volume serial number:</td>
<td>LC Position in the message.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 1, Max 3-digit</td>
<td></td>
</tr>
<tr>
<td>X'86F500'</td>
<td>OVOL</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>LB SB</td>
<td></td>
</tr>
<tr>
<td>X'870000'</td>
<td>OWN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>GV LD LO LP LS LV SD(e) SP SS SV</td>
<td></td>
</tr>
<tr>
<td>X'871000'</td>
<td>OXDJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Original expiration date</td>
<td></td>
</tr>
</tbody>
</table>

Table 16. Command SFIs (continued)
<table>
<thead>
<tr>
<th>SFI Number</th>
<th>SFI Name</th>
<th>SFI Length</th>
<th>SFI Data Type</th>
<th>Data Description</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'871800'</td>
<td>PACS</td>
<td>9</td>
<td>Binary(8)</td>
<td>PREACS</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=NO 1=YES</td>
<td></td>
</tr>
<tr>
<td>X'871E00'</td>
<td>PDA</td>
<td>9</td>
<td>Binary(8)</td>
<td>PDA state:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=Off 1=On</td>
<td></td>
</tr>
<tr>
<td>X'871E10'</td>
<td>PDAC</td>
<td>9</td>
<td>Binary(8)</td>
<td>PDA block count:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Numeric 2-255</td>
<td></td>
</tr>
<tr>
<td>X'871E30'</td>
<td>PDAL</td>
<td>9</td>
<td>Binary(8)</td>
<td>PDA log state:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=Off 1=On</td>
<td></td>
</tr>
<tr>
<td>X'871E90'</td>
<td>PDAS</td>
<td>9</td>
<td>Binary(8)</td>
<td>PDA block size:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Numeric 1-31</td>
<td></td>
</tr>
<tr>
<td>X'872000'</td>
<td>PDS</td>
<td>48</td>
<td>Character (variable length)</td>
<td>Pool description</td>
<td>LC</td>
</tr>
<tr>
<td>X'873000'</td>
<td>PDSC</td>
<td>40</td>
<td>Character (variable length)</td>
<td>Product description</td>
<td>LP SP(e)</td>
</tr>
<tr>
<td>X'874000'</td>
<td>PEND</td>
<td>9</td>
<td>Bit(8)</td>
<td>Actions pending:</td>
<td>LV SV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'80'=SCRATCH</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'40'=REPLACE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'20'=INIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'10'=ERASE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'08'=RETURN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'04'=NOTIFY</td>
<td></td>
</tr>
<tr>
<td>X'875000'</td>
<td>PID</td>
<td>14</td>
<td>Character (variable length)</td>
<td>Pool prefix</td>
<td>LC LR SR</td>
</tr>
<tr>
<td>X'876000'</td>
<td>PLN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Pool name</td>
<td>LC</td>
</tr>
<tr>
<td>X'877000'</td>
<td>PNME</td>
<td>38</td>
<td>Character (variable length)</td>
<td>Software product name</td>
<td>LP SP</td>
</tr>
<tr>
<td>X'878000'</td>
<td>PNUM</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Software product number</td>
<td>LP LV SP SV(e)</td>
</tr>
<tr>
<td>X'879000'</td>
<td>PRD</td>
<td>12</td>
<td>Binary(32)</td>
<td>Permanent read errors:</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 5-digit</td>
<td></td>
</tr>
<tr>
<td>X'87A000'</td>
<td>PRF</td>
<td>9</td>
<td>Binary(8)</td>
<td>Pool definition RACF® (A component of the Security Server for z/OS) option:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=NO 1=YES</td>
<td></td>
</tr>
<tr>
<td>X'87B000'</td>
<td>PRTY</td>
<td>12</td>
<td>Binary(32)</td>
<td>Priority:</td>
<td>LS SS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 4-digit</td>
<td></td>
</tr>
<tr>
<td>X'87C000'</td>
<td>PSFX</td>
<td>10</td>
<td>Character (fixed length)</td>
<td>Parmlib member suffix</td>
<td>LC</td>
</tr>
<tr>
<td>X'87C010'</td>
<td>PSF2</td>
<td>10</td>
<td>Character (fixed length)</td>
<td>Second parmlib member suffix</td>
<td>LC</td>
</tr>
<tr>
<td>X'87D000'</td>
<td>PSN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Pool definition system ID</td>
<td>LC</td>
</tr>
<tr>
<td>X'87DB00'</td>
<td>PTNA</td>
<td>9</td>
<td>Binary(8)</td>
<td>NOSMT action for partition entry:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=ACCEPT 1=IGNORE</td>
<td></td>
</tr>
<tr>
<td>X'87DB0C'</td>
<td>PTNL</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Location name</td>
<td>LC</td>
</tr>
<tr>
<td>X'87E000'</td>
<td>PTP</td>
<td>9</td>
<td>Binary(8)</td>
<td>Pool definition pool type:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=SCRATCH 1=RACK</td>
<td></td>
</tr>
<tr>
<td>SFI Number</td>
<td>SFI Name</td>
<td>SFI Length</td>
<td>SFI Data Type</td>
<td>Data Description</td>
<td>Subcommand</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>---------------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>X'87EB80'</td>
<td>PTSA</td>
<td>9</td>
<td>Binary(8)</td>
<td>SMT action for partition entry: 0=ACCEPT 1=IGNORE</td>
<td>LC</td>
</tr>
<tr>
<td>X'87EBA8'</td>
<td>PTT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Type of partition entry: 0=RMM 1=NORMM</td>
<td>LC</td>
</tr>
<tr>
<td>X'87EC00'</td>
<td>PTVS</td>
<td>14</td>
<td>Character (variable length)</td>
<td>Volume range start</td>
<td>LC</td>
</tr>
<tr>
<td>X'87EC08'</td>
<td>PTVL</td>
<td>14</td>
<td>Character (variable length)</td>
<td>Volume serial number, specific or generic</td>
<td>LC</td>
</tr>
<tr>
<td>X'87EC0F'</td>
<td>PTVE</td>
<td>14</td>
<td>Character (variable length)</td>
<td>Volume range end</td>
<td>LC</td>
</tr>
<tr>
<td>X'87F000'</td>
<td>PVL</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>Previous volume: 1 - 6 character</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'880000'</td>
<td>PWT</td>
<td>12</td>
<td>Binary(32)</td>
<td>Permanent write errors: Min 0, Max 5-digit</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'881000'</td>
<td>RBN</td>
<td>12</td>
<td>Binary(32)</td>
<td>Number of bin numbers in REMOTE location: Min 0, Max 6-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'881200'</td>
<td>RBYS</td>
<td>9</td>
<td>Binary(8)</td>
<td>Retain by set: 0=NO 1=YES</td>
<td>LV SV SV(e)</td>
</tr>
<tr>
<td>X'882000'</td>
<td>RCF</td>
<td>9</td>
<td>Binary(8)</td>
<td>Installation RACF support: 1=N 2=P 3=A 4=C</td>
<td>LC</td>
</tr>
<tr>
<td>X'883000'</td>
<td>RCFM</td>
<td>12</td>
<td>Character (variable length)</td>
<td>RECFM</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'884000'</td>
<td>RCK</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>Rack or bin number</td>
<td>AB AR AV CV DB DR LB LP LR LV SB SP(e) SR SV</td>
</tr>
<tr>
<td>X'886000'</td>
<td>RDTJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Last control data set extract date</td>
<td>LC</td>
</tr>
<tr>
<td>X'888000'</td>
<td>RET</td>
<td>11</td>
<td>Binary(8)</td>
<td>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</td>
<td>LS SS</td>
</tr>
<tr>
<td>X'88A000'</td>
<td>RST</td>
<td>9</td>
<td>Binary(8)</td>
<td>Rack or bin status 0=EMPTY 1=FREE 2=INUSE</td>
<td>LB LR SB SR</td>
</tr>
<tr>
<td>X'88B900'</td>
<td>RTBY</td>
<td>9</td>
<td>Binary(8)</td>
<td>Retain by: 0=VOLUME 1=SET</td>
<td>LC</td>
</tr>
<tr>
<td>X'88C000'</td>
<td>RTDJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Retention date</td>
<td>LD LV SD(e) SV</td>
</tr>
</tbody>
</table>
Table 16. Command SFIs (continued)

<table>
<thead>
<tr>
<th>SFI Number</th>
<th>SFI Name</th>
<th>SFI Length</th>
<th>SFI Data Type</th>
<th>Data Description</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'88E000'</td>
<td>RTM</td>
<td>12</td>
<td>Packed decimal</td>
<td>Last control data set extract time</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>time format</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'88E500'</td>
<td>RUB</td>
<td>9</td>
<td>Binary(8)</td>
<td>Reuse bin at 0=CONFIRMMOVE 1=STARTMOVE</td>
<td>LC</td>
</tr>
<tr>
<td>X'890000'</td>
<td>SC</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Storage class name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'891000'</td>
<td>SCRM</td>
<td>9</td>
<td>Binary(8)</td>
<td>Binary value 0=Auto 1=manual</td>
<td>LC</td>
</tr>
<tr>
<td>X'892000'</td>
<td>SCST</td>
<td>9</td>
<td>Bit(8)</td>
<td>Security class status '80'=SMF '40'=MSGOPT '20'=ERASE</td>
<td>LC</td>
</tr>
<tr>
<td>X'894000'</td>
<td>SC1</td>
<td>12</td>
<td>Binary(32)</td>
<td>Storenumber Min 1, Max 5-digit</td>
<td>LS SS SS(e)</td>
</tr>
<tr>
<td>X'895000'</td>
<td>SDTJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Movement tracking date</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'896000'</td>
<td>SEC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Security class number Min 0, Max 255</td>
<td>LC</td>
</tr>
<tr>
<td>X'898000'</td>
<td>SEQ</td>
<td>12</td>
<td>Binary(32)</td>
<td>Volume sequence Min 1, Max 4-digit</td>
<td>LV</td>
</tr>
<tr>
<td>X'89A000'</td>
<td>SG</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Storage group name</td>
<td>LD LV SD(e) SV(e)</td>
</tr>
<tr>
<td>X'89B000'</td>
<td>SID</td>
<td>16</td>
<td>Character (variable length)</td>
<td>DFSMSrmm system ID</td>
<td>LC</td>
</tr>
<tr>
<td>X'89C000'</td>
<td>SLM</td>
<td>10</td>
<td>Binary(15)</td>
<td>MAXHOLD value Min 10, Max 500</td>
<td>LC</td>
</tr>
<tr>
<td>X'89E000'</td>
<td>SMF</td>
<td>10</td>
<td>Binary(15)</td>
<td>Offset to message ID Min 0, Max 3-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'89E210'</td>
<td>SMP</td>
<td>9</td>
<td>Binary(8)</td>
<td>System-managed tape purge: 0=NO 1=YES 2=ASIS</td>
<td>LC</td>
</tr>
<tr>
<td>X'89E220'</td>
<td>SMU</td>
<td>9</td>
<td>Bit(8)</td>
<td>System-managed tape update: 20=Command 40=Scratch 80=Exits N/A</td>
<td>LC</td>
</tr>
<tr>
<td>X'89F000'</td>
<td>SOSJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Last expiration processing start date</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A0000'</td>
<td>SOSP</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Scratch procedure name</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A1000'</td>
<td>SOST</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last expiration processing start time</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A1A00'</td>
<td>SRHN</td>
<td>71</td>
<td>Character (variable length)</td>
<td>Server host name 1-to-63 alphanumeric characters including hyphen, period, and blank</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A1A30'</td>
<td>SRIP</td>
<td>23</td>
<td>Character (variable length)</td>
<td>Server IP address 1-to-15 numeric characters including period and blank</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A1A50'</td>
<td>SRPN</td>
<td>12</td>
<td>Binary (31)</td>
<td>Server number binary value</td>
<td>LC</td>
</tr>
</tbody>
</table>

Appendix A. Structured Field Introducers 99
<table>
<thead>
<tr>
<th>SFI Number</th>
<th>SFI Name</th>
<th>SFI Length</th>
<th>SFI Data Type</th>
<th>Data Description</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>X’8A1AF0’</td>
<td>SRTK</td>
<td>12</td>
<td>Binary (31)</td>
<td>Server tasks binary value</td>
<td>LC</td>
</tr>
<tr>
<td>X’8A2000’</td>
<td>SSM</td>
<td>10</td>
<td>Binary(15)</td>
<td>SMF security record type: 128-255, 42, or 0</td>
<td>LC</td>
</tr>
<tr>
<td>X’8A2500’</td>
<td>SSTY</td>
<td>9</td>
<td>Binary (8)</td>
<td>Subsystem type 0=Standard system 1=Client system 2=Server system</td>
<td>LC</td>
</tr>
<tr>
<td>X’8A3000’</td>
<td>STEP</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Step name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X’8A3800’</td>
<td>STVC</td>
<td>12</td>
<td>Binary(32)</td>
<td>Count of volumes stacked on a stacked volume</td>
<td>LV VOL SV(e)</td>
</tr>
<tr>
<td>X’8A4000’</td>
<td>SUR</td>
<td>28</td>
<td>Character (variable length)</td>
<td>Surname</td>
<td>LO</td>
</tr>
<tr>
<td>X’8A5000’</td>
<td>SYS</td>
<td>16</td>
<td>Character (variable length)</td>
<td>SMF System ID</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X’8A6000’</td>
<td>TAC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Reject type 0=ANYUSE 1=OUTPUT</td>
<td>LC</td>
</tr>
<tr>
<td>X’8A6800’</td>
<td>TLR</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>hhmmssstC, where hhmmssst is the time in hours, minutes, seconds, and tenths of seconds and C is a standard packed-decimal sign character.</td>
<td>LS SS(e)</td>
</tr>
<tr>
<td>X’8A7000’</td>
<td>TRD</td>
<td>12</td>
<td>Binary(32)</td>
<td>Temporary read errors Min 0, Max 5-digit</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X’8A7900’</td>
<td>TVXP</td>
<td>9</td>
<td>Binary(8)</td>
<td>Tape volume exit purge option: 0=RELEASE 1=EXPIRE 2=NONE</td>
<td>LC</td>
</tr>
<tr>
<td>X’8A8000’</td>
<td>TWT</td>
<td>12</td>
<td>Binary(32)</td>
<td>Temporary write errors: Min 0, Max 5-digit</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X’8A9000’</td>
<td>TYP</td>
<td>9</td>
<td>Bit(8)</td>
<td>VRS type: '80'=GDG '40'=PSEUDGDG '20'=DSNAME '10'=VOLUME '08'=NAME</td>
<td>LS SS(e)</td>
</tr>
<tr>
<td>X’8A9E00’</td>
<td>TZ</td>
<td>12</td>
<td>Binary(32)</td>
<td>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.</td>
<td>All</td>
</tr>
<tr>
<td>X’8AA000’</td>
<td>UDTJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Late update date</td>
<td>LC</td>
</tr>
<tr>
<td>X’8AB001’</td>
<td>UID</td>
<td>16</td>
<td>Character (variable length)</td>
<td>User ID. The SFI is incremented by one for each UID that is found. (X’8AB001’-X’8AB00C’)</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X’8AC000’</td>
<td>UNC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Uncatalog option: 0=N 1=Y 2=S</td>
<td>LC</td>
</tr>
<tr>
<td>SFI Number</td>
<td>SFI Name</td>
<td>SFI Length</td>
<td>SFI Data Type</td>
<td>Data Description</td>
<td>Subcommand</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>---------------</td>
<td>------------------</td>
<td>------------</td>
</tr>
<tr>
<td>X'8AD000'</td>
<td>USEC</td>
<td>12</td>
<td>Binary(32)</td>
<td>Volume use count: Min 0, Max 5-digit</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'8AE000'</td>
<td>USEM</td>
<td>12</td>
<td>Binary(32)</td>
<td>Volume usage (KB): Min 0, Max 4294967295. 4294967295 indicates that USEM must be used.</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'8AE030'</td>
<td>USE6</td>
<td>22</td>
<td>Compound (Binary(8), Factor, Binary(64), Value)</td>
<td>Volume usage, Factor: 0=bytes 1=KB 2=MB 3=GB 4=TB Value: Minimum value = 0.</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'8AE600'</td>
<td>UTC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Common Time: 0=DISABLED 1=ENABLED</td>
<td>LC</td>
</tr>
<tr>
<td>X'8AE800'</td>
<td>UTM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Late update time</td>
<td>LC</td>
</tr>
<tr>
<td>X'8AF001'</td>
<td>VAC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Volume access: 0=NONE 1=READ 2=UPDATE</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'8B0000'</td>
<td>VACT</td>
<td>9</td>
<td>Binary(8)</td>
<td>VRSMIN action: 0=FAIL 1=INFO 2=WARN 3=OFF</td>
<td>LC</td>
</tr>
<tr>
<td>X'8B0800'</td>
<td>VANX</td>
<td>9</td>
<td>Binary(8)</td>
<td>Next VRS type: 0=Undefined 1=Next 2=And</td>
<td>LS SS</td>
</tr>
<tr>
<td>X'8B0B00'</td>
<td>VCAP</td>
<td>12</td>
<td>Binary(32)</td>
<td>Volume/Media capacity</td>
<td>LV SV(e) LC</td>
</tr>
<tr>
<td>X'8B1000'</td>
<td>VCHG</td>
<td>9</td>
<td>Binary(8)</td>
<td>VRSCHANGE value: 0=INFO 1=VERIFY</td>
<td>LC</td>
</tr>
<tr>
<td>X'8B2000'</td>
<td>VDD</td>
<td>10</td>
<td>Binary(15)</td>
<td>VRS delay days: Min 0, Max 99</td>
<td>LS SS(e)</td>
</tr>
<tr>
<td>X'8B2800'</td>
<td>VDRA</td>
<td>9</td>
<td>Binary(8)</td>
<td>VRSDROP action: 0=FAIL 1=INFO 2=WARN 3=OFF</td>
<td>LC</td>
</tr>
<tr>
<td>X'8B2802'</td>
<td>VDRC</td>
<td>12</td>
<td>Binary(32)</td>
<td>VRSDROP count</td>
<td>LC</td>
</tr>
<tr>
<td>X'8B280F'</td>
<td>VDRP</td>
<td>10</td>
<td>Binary(15)</td>
<td>VRSDROP percent</td>
<td>LC</td>
</tr>
<tr>
<td>X'8B3000'</td>
<td>VDTJ</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last inventory management processing date</td>
<td>LC</td>
</tr>
<tr>
<td>X'8B4000'</td>
<td>VER</td>
<td>14</td>
<td>Character (variable length)</td>
<td>Software produce version, release, modification vvrrmm</td>
<td>LP LV SP SV(e)</td>
</tr>
<tr>
<td>X'8B5000'</td>
<td>VJBN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Primary VRS job name</td>
<td>LD LS SD(e) SS</td>
</tr>
<tr>
<td>X'8B6000'</td>
<td>VLN</td>
<td>12</td>
<td>Binary(32)</td>
<td>Number of volumes: Min 0, Max 3-digit</td>
<td>LO LP SP</td>
</tr>
<tr>
<td>X'8B7000'</td>
<td>VM</td>
<td>9</td>
<td>Binary(8)</td>
<td>VM use: 0=NO 1=YES</td>
<td>LV SV(e)</td>
</tr>
</tbody>
</table>
Table 16. Command SFIs (continued)

<table>
<thead>
<tr>
<th>SFI Number</th>
<th>SFI Name</th>
<th>SFI Length</th>
<th>SFI Data Type</th>
<th>Data Description</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'8B8000'</td>
<td>VMIN</td>
<td>12</td>
<td>Binary(32)</td>
<td>VRSMIN count value: Min 0, Max 6-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'8B9000'</td>
<td>VMV</td>
<td>16</td>
<td>Character (variable length)</td>
<td>VRS management value</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'8B9100'</td>
<td>VWMC</td>
<td>12</td>
<td>Binary(32)</td>
<td>Volume write mount count</td>
<td>LV, SV(e)</td>
</tr>
<tr>
<td>X'8B9E00'</td>
<td>VNDR</td>
<td>16</td>
<td>Character (8)</td>
<td>Vendor information</td>
<td>LV, SV(e)</td>
</tr>
<tr>
<td>X'8BA000'</td>
<td>VNME</td>
<td>52</td>
<td>Character (variable length)</td>
<td>Primary VRS name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'8BC000'</td>
<td>VOL</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>1 - 6 characters volume serial</td>
<td>AV CV GV LB LD LP LR LV SB SD SP SR SV</td>
</tr>
<tr>
<td>X'8BC200'</td>
<td>VOLT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Volume type: 0=PHYSICAL 1=LOGICAL 2=STACKED</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'8BCD00'</td>
<td>VOL1</td>
<td>14</td>
<td>Character (fixed length)</td>
<td>VOL1 label volume serial number</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'8BC300'</td>
<td>VPCT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Volume percent full</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'8BD000'</td>
<td>VRC</td>
<td>12</td>
<td>Binary(32)</td>
<td>Vital record count: Min 1, Max 5-digit</td>
<td>LS SS SS(e)</td>
</tr>
<tr>
<td>X'8BD500'</td>
<td>VREA</td>
<td>9</td>
<td>Binary(8)</td>
<td>VRSRETAIN action: 0=FAIL 1=INFO 2=WARN 3=OFF</td>
<td>LC</td>
</tr>
<tr>
<td>X'8BD502'</td>
<td>VREC</td>
<td>12</td>
<td>Binary(32)</td>
<td>VRSRETAIN count</td>
<td>LC</td>
</tr>
<tr>
<td>X'8BD50F'</td>
<td>VREP</td>
<td>10</td>
<td>Binary(15)</td>
<td>VRSRETAIN percent</td>
<td>LC</td>
</tr>
<tr>
<td>X'8BE000'</td>
<td>VRJ</td>
<td>9</td>
<td>Binary(8)</td>
<td>VRS job name: 1 or 2</td>
<td>LC</td>
</tr>
<tr>
<td>X'8BF000'</td>
<td>VRS</td>
<td>52</td>
<td>Character (variable length)</td>
<td>Vital record specification name</td>
<td>LS SS</td>
</tr>
<tr>
<td>X'8BF500'</td>
<td>VRSI</td>
<td>9</td>
<td>Binary(8)</td>
<td>Release action scratch immediate: 0=NO 1=YES</td>
<td>LS LV SS SV(e)</td>
</tr>
<tr>
<td>X'8BFA00'</td>
<td>VRSL</td>
<td>9</td>
<td>Binary(8)</td>
<td>VRSEL value: 1=NEW</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C0000'</td>
<td>VRSR</td>
<td>9</td>
<td>Binary(8)</td>
<td>VRS retained status: 0=NO 1=YES</td>
<td>LD SD SD(e)</td>
</tr>
<tr>
<td>X'8C0800'</td>
<td>VRXI</td>
<td>9</td>
<td>Binary(8)</td>
<td>Expiration date ignore: 0=NO 1=YES</td>
<td>LV LS SS SV(e)</td>
</tr>
<tr>
<td>X'8C1000'</td>
<td>VSCD</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Primary VRS subchain start date</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'8C1800'</td>
<td>VSCN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Primary VRS subchain name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'8C2000'</td>
<td>VST</td>
<td>9</td>
<td>Bit(8)</td>
<td>Volume status: '80'=MASTER '40'=SCRATCH '20'=USER '10'=INIT '08'=ENTRY</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'8C3000'</td>
<td>VTM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last inventory management VRS time</td>
<td>LC</td>
</tr>
<tr>
<td>SFI Number</td>
<td>SFI Name</td>
<td>SFI Length</td>
<td>SFI Data Type</td>
<td>Data Description</td>
<td>Subcommand</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>---------------</td>
<td>------------------</td>
<td>------------</td>
</tr>
<tr>
<td>X'8C4000'</td>
<td>VTYP</td>
<td>9</td>
<td>Binary(8)</td>
<td>Matching VRS type: 0=UNDEFINED 1=DATASET 2=SMSMC 3=VRSMV 4=DSNMV 5=DSNMC</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'8C4300'</td>
<td>WORM</td>
<td>9</td>
<td>Binary(8)</td>
<td>Volume is WORM: 0=NO 1=YES</td>
<td>LV, SV (e)</td>
</tr>
<tr>
<td>X'8C4500'</td>
<td>WWID</td>
<td>32</td>
<td>Character (24)</td>
<td>World-wide identifier</td>
<td>LV, SV (e)</td>
</tr>
<tr>
<td>X'8C5000'</td>
<td>XDC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Expiration date check: 0=NO 1=YES 2=OPERATOR</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C5D00'</td>
<td>XDRA</td>
<td>9</td>
<td>Binary(8)</td>
<td>EXPDTRDROP action: 0=FAIL 1=INFO 2=WARN 3=OFF</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C5D02'</td>
<td>XDRC</td>
<td>12</td>
<td>Binary(32)</td>
<td>EXPDTRDROP count</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C5D0F'</td>
<td>XDRP</td>
<td>10</td>
<td>Binary(15)</td>
<td>EXPDTRDROP percent</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C6000'</td>
<td>XDTJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Expiration date</td>
<td>LC LD LV SD SV</td>
</tr>
<tr>
<td>X'8C7000'</td>
<td>XTM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last inventory management expiration time</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C7800'</td>
<td>X100</td>
<td>9</td>
<td>Binary(8)</td>
<td>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</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C7801'</td>
<td>X200</td>
<td>9</td>
<td>Binary(8)</td>
<td>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</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C7802'</td>
<td>X300</td>
<td>9</td>
<td>Binary(8)</td>
<td>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</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C8000'</td>
<td>2JBN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Secondary VRS jobname mask</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'8C9000'</td>
<td>2NME</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Secondary VRS mask</td>
<td>LD SD(e)</td>
</tr>
</tbody>
</table>
Table 16. Command SFIs (continued)

<table>
<thead>
<tr>
<th>SFI Number</th>
<th>SFI Name</th>
<th>SFI Length</th>
<th>SFI Data Type</th>
<th>Data Description</th>
<th>Subcommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'8CA000'</td>
<td>2SCD</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Secondary VRS subchain start date</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'8CB000'</td>
<td>2SCN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Secondary VRS subchain name</td>
<td>LD SD(e)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Structured Field Introducers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDBIN</td>
<td>CNT ENTN MSGL MSGN RCK RSNC RTNC SVCN</td>
</tr>
<tr>
<td>ADDDATASET</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>ADDOWNER</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>ADDPRODUCT</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>ADDRACK</td>
<td>CNT ENTN MSGL MSGN RCK RSNC RTNC SVCN</td>
</tr>
<tr>
<td>ADDVOLUME</td>
<td>CLIB CNT CSG ENTN FRC FR S MSGL MSGN RCK RSNC RTNC SVCN</td>
</tr>
<tr>
<td>ADDVRS</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>CHANGEDATASET</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>CHANGEOWNER</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>CHANGEPRODUCT</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>CHANGEVOLUME</td>
<td>CLIB CSG ENTN FRC FR S MEDN MSGL MSGN RCK RSNC RTNC SVCN</td>
</tr>
<tr>
<td>CHANGEVRS</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>DELETEBIN</td>
<td>CNT ENTN MSGL MSGN RCK RSNC RTNC SVCN</td>
</tr>
<tr>
<td>DELETEDATASET</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>DELETEOWNER</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>DELETERACK</td>
<td>CNT ENTN MSGL MSGN RCK RSNC RTNC SVCN</td>
</tr>
<tr>
<td>DELETEVOLUME</td>
<td>CLIB ENTN FRC FR S MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>DELETEVRS</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>GETVOLUME</td>
<td>ENTN FRC FR S MSGL MSGN OWN RSNC RTNC SVCN VOL</td>
</tr>
<tr>
<td>LISTBIN</td>
<td>ENTN LINE LOC MIV MOV MEDN MSGL MSGN OVOL RCK RSNC RTNC SVCN TZ VOL</td>
</tr>
<tr>
<td>LISTCONTROL ACTIONS</td>
<td>ACT AST RC</td>
</tr>
<tr>
<td>LISTCONTROL CNTL</td>
<td>ACS AUD BDT BTM CDSQ CDSU CSHN CSIP CSVE DBN CDS CSDT CSTM DDT DNP DTE DTM EBIN FBP FCSP FDB FEP FKP FLB FRB FRK FSP FTP FVP FXP IPL JBDT JBTM JDS JRNS JRU LBN LCT LRD MDS MDT MRP MTM MTP NOT OPM PACS RBN RC RCF RDT RMT RUB SAT SDT SID SLM SOSD SOSP SOST SSM STM UDT UTC UTM VDT VTM XDTJ XTM X100 X200 X300</td>
</tr>
<tr>
<td>Subcommand</td>
<td>Structured Field Introducers</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td>LISTCONTROL LOCDEF</td>
<td>LDLC LDIT LDIT LDLT LDLMN LDPR RC</td>
</tr>
<tr>
<td>LISTCONTROL MNTMSG</td>
<td>MID OPL OVL RC SMI</td>
</tr>
<tr>
<td>LISTCONTROL MEDINF</td>
<td>MDNF MDRA MDRP MDRT MDRW MDRX MDTX MEDR MEDT VCAP</td>
</tr>
<tr>
<td>LISTCONTROL MOVES</td>
<td>MFR MST MTO MTY</td>
</tr>
<tr>
<td>LISTCONTROL OPENRULE</td>
<td>ORIA ORII ORAO ORIA OROR ORTP ORVE ORVL ORVS</td>
</tr>
<tr>
<td>LISTCONTROL OPTION</td>
<td>ACCT AUD BLP BKPP CATS CDS CMDD CMDO CRP DRP DSPD DSS6 ENTP FILE</td>
</tr>
<tr>
<td></td>
<td>LDD LDEV LINE LPGM LRLT LSTP MC MSGL MSGN NME OWN OXDJ RCFM RSNC RTDJ RTNC SC SG STEP SVCN SYS TZ VJBN VMSME VOL VRSR VSCP VSN CP VTP XTJ 2JBN 2NME 2SCD 2SCN</td>
</tr>
<tr>
<td>LISTCONTROL PRITION</td>
<td>PTNA PTNL PTSA PTP PTV TE PTVL PTVS</td>
</tr>
<tr>
<td>LISTCONTROL REJECT</td>
<td>GRK RC TAC</td>
</tr>
<tr>
<td>LISTCONTROL SECCLS</td>
<td>CLS ERS MSG NME RC SEC SMF</td>
</tr>
<tr>
<td>LISTCONTROL SEGLEVEL</td>
<td>CLS DNM ERS MSG NME RC SEC SMF</td>
</tr>
<tr>
<td>LISTCONTROL VLPOOL</td>
<td>ACT MEDN MOP PDS PID PLN PRF PSN PTP SCRM XDC</td>
</tr>
<tr>
<td>LISTDATASET</td>
<td>ABND BLKC BLKS BLKT CD TJ CBN CLS CPGM CTLG CMT DC</td>
</tr>
<tr>
<td></td>
<td>DD DEV DL RJ DL TD DL WJ DP CT DSEQ DSN DSS6 ENTP FILE LDD LDEV LINE LPGM LRLT LSTP MC MSGL MSGN NME OWN OXDJ RCFM RSNC RTDJ RTNC SC SG STEP SVCN SYS TZ VJ BN VMSME VOL VRSR VSCP VSN CP VTP XTJ 2JBN 2NME 2SCD 2SCN</td>
</tr>
<tr>
<td>LISTOWNER</td>
<td>ADL DPT EML EMN EMU ENTP ETL FOR ITL LINE MSGL MSGN</td>
</tr>
<tr>
<td></td>
<td>OWN RSNC RTNC SUR SVCN TZ VLN</td>
</tr>
<tr>
<td>LISTPRODUCT</td>
<td>ENTP FCD LINE MSGL MSGN OWN PDS PNME PNUM RCK RSNC RTNC SVCN TZ VLN VOL</td>
</tr>
<tr>
<td>LISTRACK</td>
<td>ENTP LINE LOC MEDN MSGL MSGN PID RCK RSNC RST RTNC SVCN VOL</td>
</tr>
<tr>
<td>LISTVOLUME</td>
<td>ACN ACT ADT J AT TM AVL BIN BMN CD TJ CBN CLS CPGM CTM CTNR</td>
</tr>
<tr>
<td></td>
<td>DBN DBMN DEN DESC DEST DL RJ DL WJ DSC DSN DSR D S T E NTP FCD HLOC HLOT INTR KEL1 KEL2 KEM1 KEM2 LBL LCID LDEV LINE LOAN LOC LOCT LVC LVN MDNF MDRX MDTX MENDA MEDN MEDR MEDT MOV MSLG MSGN MYS NLLOC NL NME NML OAC OB MN OB N OCE OLOC OLOT OWN OXDJ PEN PNUM PRD PVL PWT RBYS RCK RSNC RTDJ RTNC SD TJ SEQ SG STVC SVCN TRD TWT TZ UID01...UID12 USEC USEM USEP USEQ VAC VCAP VER VMS VMIN VNDR VOL VOLT VOL1 VPCT VRSI VRSI VST VWMC WORM WWID XDTJ</td>
</tr>
<tr>
<td>LISTVRS</td>
<td>DDTJ DESC DL RJ ENTP LINE LOC MSGL MSGN NVRS OWN PR TY RET RSNC RTNC SC1 SVCN TLR TYP TZ VAXX VDD VJBN VRC VRS VRSI VRSX</td>
</tr>
<tr>
<td>SEARCHBIN</td>
<td>CONT ENTP LINE LOC MEDN MIV MOV MSGL MSGN OVL RCK RSNC RTNC SVCN TZ VOL</td>
</tr>
<tr>
<td>SEARCHDATASET</td>
<td>CD T J CONT CTM DSN ENTP FILE KEYF KEYT LINE MSGL MSGN OWN OXDJ RSNC RTNC SVCN VOL XDTJ</td>
</tr>
<tr>
<td>SEARCHDATASET(EXPAND=YES)</td>
<td>The same SFIs as the LISTDATASET subcommand.</td>
</tr>
<tr>
<td>Subcommand</td>
<td>Structured Field Introducers</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>SEARCHOWNER</td>
<td>ADL CONT DPT EML EMN EMU ETL FOR ITL OWN SUR TZ VLN</td>
</tr>
<tr>
<td>SEARCHPRODUCT</td>
<td>CONT ENTN FCD LINE MSGN OWN PNME PNUM RSNC RTNC SVCN VER VLN VOL</td>
</tr>
<tr>
<td>SEARCHPRODUCT(EXPAND=NO)</td>
<td>The same SFIs as the LISTPRODUCT subcommand.</td>
</tr>
<tr>
<td>SEARCHRACK</td>
<td>CONT ENTN LINE LOC MEDN MSGN PID RCK RSNC RST RTNC SVCN VOL</td>
</tr>
<tr>
<td>SEARCHVOLUME</td>
<td>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 MSGN OWN PEND RCK RSNC RTDJ RTNC SEQ SVCN TYPF TYPF VCAP VOL VST XDTJ</td>
</tr>
<tr>
<td>SEARCHVOLUME(EXPAND=NO)</td>
<td>The same SFIs as the LISTVOLUME subcommand.</td>
</tr>
<tr>
<td>SEARCHVRS</td>
<td>CONT DDTJ ENTN LINE LOC MSGN NVRS OWN PRTY RET RSNC RTNC SVCN VANX VJBN VRS VRSI VRXI</td>
</tr>
<tr>
<td>SEARCHVRS(EXPAND=NO)</td>
<td>The same SFIs as the LISTVRS subcommand.</td>
</tr>
</tbody>
</table>
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 OD ++ 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_XSUBCMADDR DS A ++ XSUBCMADDR |
| 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 |
| MYPL EQU **MYPL ++ LENGTH OF PLIST |
```

Figure 53. Mapping of the Parameter List Using the List Form of EDGXCI

© Copyright IBM Corp. 1992, 2009
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**
  - **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.
    - **NO**
      - Indicates that a DSECT statement should not be generated.

- **LIST=**
  - **YES**
  - **NO**

  An optional parameter that specifies whether the macro expansion is printed. The default is LIST=YES.

  - **LIST=**
    - **YES**
      - Indicates to print the expansion.
    - **NO**
      - Indicates do not print the expansion.

- **TITLE=**
  - **YES**
  - **NO**

  An optional parameter that specifies whether the macro title is printed. The default is TITLE=YES.

  - **TITLE=**
    - **YES**
      - Indicates to print the 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

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Dec</th>
<th>Hex</th>
<th>Type</th>
<th>Len</th>
<th>Name(Dim)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>0</td>
<td>@</td>
<td>STRUCTURE</td>
<td>*</td>
<td>XSF_OUTBUF</td>
<td>Output Buffer</td>
</tr>
<tr>
<td>0 (0)</td>
<td>0</td>
<td>@</td>
<td>SIGNED</td>
<td>4</td>
<td>XSF_OUTBUF_BUFLNG</td>
<td>Output Buffer Length</td>
</tr>
<tr>
<td>4 (4)</td>
<td>4</td>
<td>@</td>
<td>SIGNED</td>
<td>4</td>
<td>XSF_OUTBUF_RQDLNG</td>
<td>Required Buffer Length</td>
</tr>
<tr>
<td>8 (8)</td>
<td>8</td>
<td>@</td>
<td>SIGNED</td>
<td>4</td>
<td>XSF_OUTBUF_DATALNG</td>
<td>Length of Output Data</td>
</tr>
<tr>
<td>12 (C)</td>
<td>12</td>
<td>@</td>
<td>CHARACTER</td>
<td>*</td>
<td>XSF_OUTBUF_FIELDS</td>
<td>Start of Structured Fields</td>
</tr>
</tbody>
</table>

Structured Field Introducers for Structured Fields

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Dec</th>
<th>Hex</th>
<th>Type</th>
<th>Len</th>
<th>Name(Dim)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (0)</td>
<td>0</td>
<td>@</td>
<td>STRUCTURE</td>
<td>*</td>
<td>XSF_SFI</td>
<td>Structured Field Introducers</td>
</tr>
<tr>
<td>0 (0)</td>
<td>0</td>
<td>@</td>
<td>CHARACTER</td>
<td>8</td>
<td>XSF_SFI_HD</td>
<td></td>
</tr>
<tr>
<td>0 (0)</td>
<td>0</td>
<td>@</td>
<td>SIGNED</td>
<td>2</td>
<td>XSF_SFI_LENGTH</td>
<td>Length</td>
</tr>
<tr>
<td>2 (2)</td>
<td>2</td>
<td>@</td>
<td>CHARACTER</td>
<td>3</td>
<td>XSF_SFI_ID</td>
<td>Identifier</td>
</tr>
<tr>
<td>2 (2)</td>
<td>2</td>
<td>@</td>
<td>CHARACTER</td>
<td>2</td>
<td>XSF_SFI_IDVAL</td>
<td>Identifier value</td>
</tr>
<tr>
<td>4 (4)</td>
<td>4</td>
<td>@</td>
<td>CHARACTER</td>
<td>1</td>
<td>XSF_SFI_IDQUAL</td>
<td>Identifier qualifier</td>
</tr>
<tr>
<td>5 (5)</td>
<td>5</td>
<td>@</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_TYPE</td>
<td>Type</td>
</tr>
<tr>
<td>6 (6)</td>
<td>6</td>
<td>@</td>
<td>CHARACTER</td>
<td>1</td>
<td>*</td>
<td>Reserved</td>
</tr>
<tr>
<td>7 (7)</td>
<td>7</td>
<td>@</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_DTYPE</td>
<td>Data type</td>
</tr>
<tr>
<td>8 (8)</td>
<td>8</td>
<td>@</td>
<td>CHARACTER</td>
<td>*</td>
<td>XSF_SFI_DATA</td>
<td>Start of data</td>
</tr>
</tbody>
</table>

Compound SFI definition

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Dec</th>
<th>Hex</th>
<th>Type</th>
<th>Len</th>
<th>Name(Dim)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 (8)</td>
<td>8</td>
<td>@</td>
<td>STRUCTURE</td>
<td>14</td>
<td>XSF_SFI_COMPTYPE1</td>
<td>Compound section</td>
</tr>
<tr>
<td>8 (8)</td>
<td>8</td>
<td>@</td>
<td>CHARACTER</td>
<td>6</td>
<td>XSF_SFI_COMPDATA</td>
<td></td>
</tr>
<tr>
<td>8 (8)</td>
<td>8</td>
<td>@</td>
<td>CHARACTER</td>
<td>6</td>
<td>XSF_SFI_COMPHDR</td>
<td>Compound Hdr</td>
</tr>
<tr>
<td>8 (8)</td>
<td>8</td>
<td>@</td>
<td>CHARACTER</td>
<td>6</td>
<td>XSF_SFI_COMPENT</td>
<td>Compound Entry</td>
</tr>
<tr>
<td>8 (8)</td>
<td>8</td>
<td>@</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_COMPTYPE</td>
<td>Compound type</td>
</tr>
<tr>
<td>9 (9)</td>
<td>9</td>
<td>@</td>
<td>CHARACTER</td>
<td>3</td>
<td>XSF_SFI_FIELD1</td>
<td></td>
</tr>
<tr>
<td>9 (9)</td>
<td>9</td>
<td>@</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_LEN1</td>
<td>Length of first field</td>
</tr>
<tr>
<td>10 (A)</td>
<td>10</td>
<td>@</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_DTYP1</td>
<td>Type of first field</td>
</tr>
<tr>
<td>11 (8)</td>
<td>11</td>
<td>@</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_FACTOR</td>
<td>Factor for second field</td>
</tr>
<tr>
<td>12 (C)</td>
<td>12</td>
<td>@</td>
<td>CHARACTER</td>
<td>2</td>
<td>XSF_SFI_FIELD2</td>
<td></td>
</tr>
<tr>
<td>12 (C)</td>
<td>12</td>
<td>@</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_LEN2</td>
<td>Length of second field</td>
</tr>
<tr>
<td>13 (D)</td>
<td>13</td>
<td>@</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_DTYP2</td>
<td>Type of second field</td>
</tr>
<tr>
<td>14 (E)</td>
<td>14</td>
<td>@</td>
<td>CHARACTER</td>
<td>8</td>
<td>XSF_SFI_COMPVAL</td>
<td>The Value</td>
</tr>
</tbody>
</table>
### Table 19. Constants for EDGSFI

<table>
<thead>
<tr>
<th>Len</th>
<th>Type</th>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HEX</td>
<td>00</td>
<td>XSF_SFI_DTYPE_UNDEF</td>
<td>Undefined data</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>01</td>
<td>XSF_SFI_DTYPE_CHAR_FIX</td>
<td>n-byte character</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>02</td>
<td>XSF_SFI_DTYPE_BITFLAG</td>
<td>1-byte bit flag byte (8 bits)</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>03</td>
<td>XSF_SFI_DTYPE_BIN8</td>
<td>1-byte (hex) value</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>04</td>
<td>XSF_SFI_DTYPE_BIN15</td>
<td>2-byte hex value</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>05</td>
<td>XSF_SFI_DTYPE_BIN31</td>
<td>4-byte hex value</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>06</td>
<td>XSF_SFI_DTYPE_BIN64</td>
<td>8-byte hex value</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>07</td>
<td>XSF_SFI_DTYPE_CHAR_VAR</td>
<td>Variable length character</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>08</td>
<td>XSF_SFI_DTYPE_COMPOUND</td>
<td>Compound SFI</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>09</td>
<td>XSF_SFI_DTYPE_JDATE</td>
<td>4-byte packed decimal date yyyydddcc - c is sign</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>0A</td>
<td>XSF_SFI_DTYPE_TIME</td>
<td>4-byte packed decimal time hhmmssccc - c is sign</td>
</tr>
</tbody>
</table>

### Compound Types (XSF_SFI_CompType)

<table>
<thead>
<tr>
<th>Len</th>
<th>Type</th>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HEX</td>
<td>00</td>
<td>XSF_SFI_COMPTYPE_UNDEF</td>
<td>Undefined type</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>01</td>
<td>XSF_SFI_COMPTYPE_FACTOR</td>
<td>Factored type</td>
</tr>
</tbody>
</table>

### Factors (XSF_SFI_Factor)

<table>
<thead>
<tr>
<th>Len</th>
<th>Type</th>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HEX</td>
<td>00</td>
<td>XSF_SFI_FACTOR_BYTES</td>
<td>Value is in Bytes</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>01</td>
<td>XSF_SFI_FACTOR_KB</td>
<td>Value is in KBytes</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>02</td>
<td>XSF_SFI_FACTOR_MB</td>
<td>Value is in MBytes</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>03</td>
<td>XSF_SFI_FACTOR_GB</td>
<td>Value is in GBytes</td>
</tr>
<tr>
<td></td>
<td>HEX</td>
<td>04</td>
<td>XSF_SFI_FACTOR_TB</td>
<td>Value is in TBytes</td>
</tr>
</tbody>
</table>

### 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.

```plaintext
* ****************************************************************************
* ** 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_xxx_GRP_LENGTH
- XSF_SFI_ID_Exxx and XSF_Exxx_GRP_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.

* ********************************************************************
* ** 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.

* ********************************************************************
* ** 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.
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.

Description of the Contents of an Output Buffer

The first line of the output buffer shown in Figure 59 shows:

0000 00008400080260 00000001A82A0 0000007D9D4D4E4
0020 E2C5D948C6C9C5D3 C48E3C5E2E3000E 88C0000000001E506 D3F0F0F1000F8700
0040 0000007D9D4D4E4 E2C5D9000C8A9E00 000005FFFF9D9000 0CB13000000000000
0060 05320F000C81A000 0000A068226F000 C8330000000000000 00000100802600
0080 0000000000000000 0000000000000000 0000000000000000 0000000000000000
00FFC 0000000000000000 0000000000000000 0000000000000000 0000000000000000
00FFE 0000000000000000 0000000000000000 0000000000000000 0000000000000000

Figure 59. Hexadecimal Representation of the Contents of an Output Buffer

- Three 4-byte length fields:
  - This is the length you specified for the output buffer.
  - 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.
  - 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:
  - 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 000000710008260 00000000001B82A0 00000007D6E6D5C
0020 D9D6D5C54BC6C9C5 D3C44BE3C5E2E300 0E88C000000001E5 D6D3F0F0F1001087

• Data Set Name structured field
  001B82A0000000007 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
  00E8BC0000000001 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 0E88C000000001E5 D6D3F0F0F1001087
0040 0000000007D6E6D5C5D9D6D5C5000C81 3000000009199711 7C000CB1A0000000

• 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 0000000007D6E6D5C50000C81 3000000009199711 7C000CB1A0000000
0060 0A0815270C000CB3 3000000005000000 0100080260800000 000000000000000

• 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.

2. Base the XSF_SFI definition in macro EDGXSF as shown in Figure 61 on the address of XSF_OUTBUF_FIELDS.

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_DATALNG.

6. Determine the type of data you are processing, by using the value in XSF_SFI_DTYPE.

7. Obtain the length of the data that starts at XSF_SFI_DATA, by subtracting XSF_SFI_LEN from the structured field length at XSF_SFI_LENGTH in the output buffer.

8. Move to the end of the SFI by adjusting the pointer. In this example, when your pointer is at offset x'00000071' into the output buffer, there are two indicators that you are done with the contents of the buffer:
   - You are looking at the End DATASET group SFI.

---

**Figure 60. Output Buffer Definition**

**Figure 61. SFI Definition**
**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:
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 103
2NME ('8C9000') - Secondary VRS 103
2SCD ('8CA000') - Secondary VRS 104
2SCN ('8CB000') - Secondary VRS 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
SFIs for 62
subcommand abbreviation 2
ADDDATASET
SFIs for 62
subcommand abbreviation 2
ADDOwnerer
SFIs for 62
subcommand abbreviation 2
ADDPRODUCT
SFIs for 62
subcommand abbreviation 2
ADDRACK
SFIs for 62
subcommand abbreviation 2
address line SFI 87
ADDVOLUME
SFIs for 62
subcommand abbreviation 2
ADDVRS
SFIs 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 Count SFI 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 ('80D300') - 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 88
CATRETPD Retention Period SFI 88
CATSYSID Value 88
CDS ('812000') - Control Data Set Identifier 88
CDSQ ('812900') - Control Data Set ENQ SFI 88
CSG ('819000') - Current Storage Group 88
CSIP ('819250') - Client IP address SFI 89
CSN ('819200') - Client/server host name SFI 89
Current label version SFI 93
Current Library Name SFI 88
Current Storage Group SFI 88
MEDR ('857000') - Media Recording Format 94
MEDI ('858000') - Media Type 94
memory size limitation 30
Message Line SFI 86
Message Number SFI 86
message retrieval tool, LookAt 94
Message SFI 86
Message Text Case SFI 95
Message Variable SFI 86
MFR ('859000') - Source Location 94
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
MOV ('85A900') - 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 areas 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 ('868000') - 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
OBM ('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
OR ('86E8A9') - Input ignore condition 96
ORIR ('86E8B8') - Input reject condition 96
OROA ('86E8A0') - Output action 96
OROI ('86E8A1') - Output ignore condition 96
OROR ('86EA18') - Output reject condition 96
ORTP ('86EA08') - Type of open rule entry 96
ORVE ('86E8F6') - Volume range end 96
ORVL ('86E8F5') - Volume serial number 96
ORVS ('86E8F0') - 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
OVL ('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
Persistent Read 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 ('87B800') - Priority 97
PSF2 ('87C010') - Second Parmlib Member Suffix 97
PSFX ('87C000') - Parmlib Member Suffix 97
PSN ('87D000') - Pool Definition System ID 97

126

z/OS V1R11.0 DFSMSrmm Application Programming Interface
PTNA ('87DB00') - NOSMT action for partition entry 97
PTNL ('87DB0C') - Location name 97
PTP ('87E000') - Pool Definition Pool Type 97
PTSA ('87EB00') - SMT action for partition entry 98
PTTP ('87EBA0') - 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 98
PVL ('880000') - Previous Volume 98
PVL ('880000') - Permanent Write 98

RACK ('88A000') - Storage Class 99
RCF ('882000') - Installation RACF 98
RDJ ('886000') - Last Control Data Set Extract Date 98
RCFM ('883000') - Record Format 98
RCK ('884000') - Rack Number or Bin 98
RDN ('885000') - Number of Bin Numbers 98
RDTJ ('886000') - Last Control Data Set Extract Date 98
RSNC ('402000') - Reason Code 85
RSN ('884000') - Reason Code 85
RTP ('885000') - Rack or Bin Status 98
RBYS ('881200') - Retain by set 98
RCF ('882000') - Installation RACF Support 98
RCL ('883000') - Record Format 98
RSN ('884000') - Reason Code 85
RSN ('884000') - Reason Code 85

Rack Number or Bin Number SFI 98
Rack Status SFI 98
RBN ('881000') - Number of Bin Numbers in REMOTE 98
RBN ('881000') - Number of Bin Numbers 98
RBYS ('881200') - Retain by set 98
RCF ('882000') - Installation RACF Support 98

RA ('888000') - Retention Type 98
Release Action Scratch Immediate SFI 102
releasing all resources 39
Required label version 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 ('88E500') - 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
SCRM ('891000') - Scratch mode 99
SCST ('892000') - Security Class Status 99
SDT ('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 IP address SFI 99
Server IP address SFI 99
Server IP address 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 58
single parameter list, single token area 36
SLM ('89C000') - MAXHOLD Value 99
SMF audit record number SFI 87
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 format 83
STV ('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 EDGXC1 5
SYS ('8A5000') - SMS 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
Time Last Referenced 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

Index 127
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: mhvrdfs@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