Contents

Figures .................................. v
Tables ................................... vii

About this document ..................... ix
Required product knowledge ............ ix
z/OS information .......................... ix
Notational conventions .......................... ix
  How to read syntax diagrams .............. x
  How to abbreviate commands and operands .......................... xii
  How to use continuation characters .............. xii
  Delimiters ............................. xii
  Character sets ............................ xii

How to send your comments to IBM .. xv
If you have a technical problem ............ xv

Summary of changes ...................... xvii
z/OS Version 2 Release 1 summary of changes xvii

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 ................ 5
    EDGXCI return and reason codes .......... 9
    EDGXCI example .................. 12

Chapter 2. Using the object-oriented DFSMSrmm application programming interface using C++ ........ 15
  DFSMSrmm high level language API classes .......... 20
    C++ classes .......................... 20
    Java class .......................... 20
  DFSMSrmm API methods .................. 20
    Java methods .......................... 21
  Receiving extensible mark up language (XML) output data in the XML output buffer .......... 22

Chapter 3. Using the DFSMSrmm application programming interface with web services ........ 25
  Sample Java web service client .......... 27
  Using persistence and parallel processing .......... 28
  Defining how and when authentication is done .......... 28

Chapter 4. Using the DFSMSrmm application programming interface using assembler language ........ 29
  Obtaining resources .......................... 29
  Specifying TSO subcommand input in the EDGXCI macro .......................... 29
  Using the CONTINUE operation in the EDGXCI macro .......................... 29
  Requesting multiple resources for SEARCH subcommands .......................... 30
  Using parameter lists to pass information to the DFSMSrmm API .......................... 31
    Coding a single parameter list, single token area .......... 32
    Coding a single parameter list, multiple token areas .......................... 34
    Coding multiple parameter lists, single token area .......................... 36
    Coding multiple parameter lists, multiple token areas .......................... 37
  Specifying the option to free a resource .......... 38
  Specifying the option to release a resource .......... 39

Chapter 5. Using an alternative interface to the DFSMSrmm application programming interface ........ 41
  Parameter list to call EDGXHINT .......................... 42
  Interface structure to pass the parameter list to EDGXHINT .......................... 43
  Communication with the API .......................... 43
    Define the API .......................... 43
    Start API communication .......... 44
    Issue a request .......................... 44
    Continue a request .......... 44
    End a request .......... 45
    End API communication .......... 45
  Return and reason codes using EDGXHINT .......... 45

Chapter 6. Processing the output data in the output buffer ........ 47
  Description of structured fields .......................... 47
  Requesting structured field introducer data format .......... 48
    Requesting line format .......... 48
    Requesting field format .......... 49
  Requesting types of output .......... 51
    Requesting standard output .......... 51
    Requesting expanded output .......... 52
  Accessing return and reason codes .......... 54
  Accessing messages and message variables .......... 54
  Interpreting date format and time format .......... 54
  Using different time zones .......... 55
  Identifying structured field introducers .......... 55
    Begin and End Resource groups .......... 56
    System return and reason code structured field introducers .......... 57
Figures

1. EDGXCI macro syntax diagram .................. 5
2. Sample job control language (JCL) for prelink step ............................................. 16
3. Sample JCL for requesting LISTVOLUME information ............................................. 17
4. C++ code example for writing XML output to a file ................................................. 22
5. XMLFILE output file ........................................... 23
6. Example of specifying the DFSMSrmm API subcommand ........................................... 29
7. Example of specifying the RMM TSO subcommand ....................................................... 29
8. Single parameter list, single token area .............................................................. 33
9. Single parameter list, multiple token areas ............................................................ 35
10. Releasing all resources .............................................. 36
11. Multiple parameter lists, single token area ........................................................... 37
12. TOKEN= specified on EDGXCI ............................................................................. 39
13. TOKEN= not specified on EDGXCI ......................................................................... 39
14. Binding a C++ program for use of EDGXHINT ....................................................... 41
15. C/C++ sample code for an interface struct ........................................................... 43
16. Issue a TSO subcommand using EDGXHINT ......................................................... 44
17. Example of list type of output using OUTPUT=LINES ........................................... 49
18. Example of output using OUTPUT=FIELDS ......................................................... 50
19. Example of search type of output using EXPAND=NO ........................................... 51
20. Example of search type of output using OUTPUT=FIELDS, EXPAND=YES .......... 52
21. Message and message variable structured fields .................................................... 54
22. Begin and End Resource group SFI sequence ....................................................... 56
23. Begin and End Resource group SFI pairs ............................................................. 56
24. Begin and End Resource group SFI pairs for subgroups ........................................ 56
25. System return and reason codes ............................................................. 57
26. Structured field introducers for messages and message variables ............................. 57
27. Message group with the CONT SFI ......................................................................... 57
28. Formatted lines ..................................................................................................... 59
29. Structured field introducers for ADDVOLUME with OUTPUT=FIELDS .................... 59
30. SFIs for CHANGEVOLUME with OUTPUT=FIELDS ............................................. 60
31. Structured field introducers for GETVOLUME with OUTPUT=FIELDS .................... 60
32. CONTINUE example, first output buffer ................................................................ 75
33. CONTINUE example, second output buffer ......................................................... 75
34. CONTINUE example, third (Last) output buffer .................................................... 76
35. Mapping of the parameter list using the list form of EDGXCI .................................... 103
36. Hexadecimal representation of the contents of an output buffer ............................... 109
37. Output buffer definition ....................................................................................... 111
38. SFI definition ...................................................................................................... 111
<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Character sets</td>
<td>xiii</td>
</tr>
<tr>
<td>2</td>
<td>Special characters used in syntax</td>
<td>xiii</td>
</tr>
<tr>
<td>3</td>
<td>RMM TSO subcommands</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Return and reason codes for the EDGXCI macro</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>DFSMSrmm API command C++ classes</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>DFSMSrmm API command Java class</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>DFSMSrmm API C++ methods</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>DFSMSrmm API Java methods</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>Return codes and reason codes issued when you specify OPERATION=CONTINUE</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>Types of parameter lists</td>
<td>31</td>
</tr>
<tr>
<td>11</td>
<td>Parameter list for a call of EDGXHINT</td>
<td>42</td>
</tr>
<tr>
<td>12</td>
<td>Message related structured field introducers</td>
<td>58</td>
</tr>
<tr>
<td>13</td>
<td>Begin and End Resource group structured field introducers</td>
<td>78</td>
</tr>
<tr>
<td>14</td>
<td>Reason and return code structured field introducers</td>
<td>79</td>
</tr>
<tr>
<td>15</td>
<td>Message structured field introducers</td>
<td>80</td>
</tr>
<tr>
<td>16</td>
<td>Command structured field introducers</td>
<td>81</td>
</tr>
<tr>
<td>17</td>
<td>Structured field introducers by subcommand</td>
<td>99</td>
</tr>
<tr>
<td>18</td>
<td>Structure XSF_OUTBUF</td>
<td>105</td>
</tr>
</tbody>
</table>
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 15 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 25 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 29 for guidelines for using the application programming interface.
- Chapter 6, “Processing the output data in the output buffer,” on page 47 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 113.

Required product knowledge

To use this document effectively, you should be familiar with:
- The RMM TSO subcommand and operands
- Macros to communicate between programs

z/OS information

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

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

To find the complete z/OS library, including the z/OS Information Center, see z/OS Internet Library (http://www.ibm.com/systems/z/os/zos/bkserv/).

Notational conventions

This section explains the notational conventions used in this document.
How to read syntax diagrams

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

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

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

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

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

- An arrow returning to the left above a keyword or value on the main path line means that the keyword or value can be repeated. The comma means that each keyword or value must be separated from the next by a comma.
An arrow returning to the left above a group of keywords or values means more than one can be selected, or a single one can be repeated.

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

If a keyword or value can be abbreviated, the abbreviation is discussed in the text associated with the syntax diagram.

If a diagram shows a character that is not alphanumeric (such as parentheses, periods, commas, and equal signs), you must code the character as part of the syntax. In this example, you must code `KEYWORD=(001,0.001)`.

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

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

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

Notes:
1. An example of a syntax note.

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

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

1ST_KEYWORD, 2ND_KEYWORD, 3RD_KEYWORD

The following is an example of a syntax diagram.

DELETEOWNER owner_ID

DO owner_ID

newowner

(1)

NEWOWNER(new_owner_id)

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 on page xiii and Table 2 on page xiii. Table 2 on page xiii 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, Numeric</td>
</tr>
<tr>
<td></td>
<td>Capital A through Z, 0 through 9</td>
</tr>
<tr>
<td>National</td>
<td>&quot;At&quot; sign, Dollar sign, Pound sign</td>
</tr>
<tr>
<td>(See note)</td>
<td>@ (Characters that can be represented by hexadecimal X'7C', X'5B', and X'7B')</td>
</tr>
<tr>
<td>Special</td>
<td>Comma, Period, Slash, Apostrophe, Left parenthesis, Right parenthesis, Asterisk, Ampersand, Plus sign, Hyphen, Equal sign, Blank</td>
</tr>
<tr>
<td>EBCDIC text</td>
<td>EBCDIC printable character set</td>
</tr>
</tbody>
</table>

**Note:** The system recognizes the following hexadecimal representations of the U.S. National characters; @ as X'7C'; $ as X'5B'; and # as X'7B'. In countries other than the U.S., the U.S. National characters represented on terminal keyboards might generate a different hexadecimal representation and cause an error. For example, in some countries the $ character may generate a X'4A'.

### Table 2. Special characters used in syntax

<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</em> //ddname DD* RESTART* on the JOB statement</td>
</tr>
<tr>
<td>(blank)</td>
<td>To enclose specified parameter values which contain special characters</td>
</tr>
<tr>
<td></td>
<td>To delimit fields</td>
</tr>
</tbody>
</table>
How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:
1. Send an email to mhvrdfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   US
4. Fax the comments to us, as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:
• Your name and address.
• Your email address.
• Your telephone or fax number.
• The publication title and order number:
  z/OS V2R1.0 DFSMSrmm Application Programming Interface
  SC23-6872-00
• The topic and page number that is related to your comment.
• The text of your comment.

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

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

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:
• Contact your IBM service representative.
• Call IBM technical support.
• Visit the IBM Support Portal at z/OS support page (http://www.ibm.com/systems/z/support/)
Summary of changes

z/OS Version 2 Release 1 summary of changes

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- z/OS Planning for Installation
- z/OS Introduction and Release Guide
- z/OS Summary of Message and Interface Changes
- z/OS Migration
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 structured field introducers (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 15
- Chapter 3, “Using the DFSMSrmm application programming interface with web services,” on page 25

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. “EDGXCI example” on page 12 provides 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 103 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 structured field introducer data format” on page 48. You can also request standard output or expanded output as described in “Requesting types of output” on page 51.

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>ADDDATASET</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>CHANGEVOLUME</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>DELETEPRODUCT</td>
<td>DP</td>
<td>Delete software product 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>
</tbody>
</table>

Refer to [z/OS DFSMSrmm Managing and Using Removable Media](#) for details on these subcommands.

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

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

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

EDGXCI: Calling the DFSMSrmm interface

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

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

EDGXCI environment

The requirements for the caller are:

Minimum authorization
Non-APF authorized, problem state and key (0-8).

Dispatchable unit mode
Task

Cross memory mode
PASN=HASN=SASN

AMODE
31-bit

ASC mode
Primary

Interrupt status
Enabled for I/O and external interrupts

Locks
The caller must not be locked.

Control parameters
Control parameters must be in the primary address space.

EDGXCI programming requirements

The caller must load the DFSMSrmm API module, EDGXAPI, prior to using the execute or standard form of EDGXCI. The caller must delete EDGXAPI when the DFSMSrmm API is no longer needed.
The caller should also use the EDGXSF macro to define the structured fields that are used in the output.

See Appendix C, “DFSMSrmm application programming interface mapping macros,” on page 103 for a complete description of the EDGXCI and EDGXSF macros.

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

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

**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**

![Diagram of EDGXCI macro syntax]

**EDGXCI parameters**

You can specify these parameters:

1. `SUBCMDADDR=subcmdaddr`, `OUTBUFADDR=bufaddr`
2. `OUTPUT=FIELDS`, `EXPAND=YES`, `MULTI=NO`
3. `OUTPUT=LINES`, `EXPAND=NO`, `MULTI=YES`
4. `APIADDR=apiaddr`
5. `OPERATION=BEGIN`, `OUTBUFADDR=bufaddr`, `TOKEN=token`
6. `OPERATION=CONTINUE`, `OUTBUFADDR=bufaddr`, `TOKEN=token`
7. `OPERATION=RELEASE`, `TOKEN=token`
8. `OPERATION=ENDALL`
9. `RETCODE=retcode`, `RSNCODE=rsncode`
10. `PLISTVER=IMPLIED_VERSION`
11. `PLISTVER=MAX`, `PLISTVER=0`
12. `MF=S`, `MF=(L`, `list addr`, `attr`, COMPLETE)}
13. `MF=(E`, `list addr`, `NOCHECK`, COMPLETE), `MF=(M`, `list addr`, `NOCHECK`, COMPLETE)

**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*
**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=NO**

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

**EXPAND=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**

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.

```plaintext
$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).

```plaintext
attr
```

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

**COMPLETE**

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

**NOCHECK**

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

**MULTI=NO**

**MULTI=YES**

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

**MULTI=NO**

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

**MULTI=YES**

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

**OPERATION=BEGIN**

**OPERATION=CONTINUE**

**OPERATION=RELEASE**

**OPERATION=ENDALL**

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

**OPERATION=BEGIN**

   Specify BEGIN to start a new subcommand.

**OPERATION=CONTINUE**

   Specify CONTINUE to continue the current subcommand.

**OPERATION=RELEASE**

   Specify when you want the token and all its associated resources to be released.

**OPERATION=ENDALL**

   Specify when you want to end all operations by releasing all tokens and all resources.

```plaintext
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>Meaning: Success.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Refer to the action provided with the specific reason code.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Meaning: EDGXCI command is successfully completed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None required.</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>Meaning: There is more output waiting to be given to you.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: 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>Meaning: Program error. An exception condition has been encountered, but the operation you requested was completed. The output results might not be acceptable to you.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Refer to the action provided with the specific reason code.</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>02</td>
<td>Meaning: There is nothing to CONTINUE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None required.</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>—</td>
<td>Meaning: 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>Action: Refer to the action provided with the specific reason code.</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>02</td>
<td>Meaning: Required token is missing.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: You need to use TOKEN=token</td>
<td></td>
</tr>
</tbody>
</table>

EDG3900I

EDG3901I

EDG3902E
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>108</td>
<td>04</td>
<td><strong>Meaning</strong>: Required address of the input subcommand is missing. <strong>Action</strong>: You need to use SUBCMDADDR=\texttt{subcmdaddr}</td>
<td>EDG3903E</td>
</tr>
<tr>
<td>108</td>
<td>06</td>
<td><strong>Meaning</strong>: Required address of your output buffer is missing. <strong>Action</strong>: Use OUTBUFADDR=\texttt{outbufaddr} to specify the parameter.</td>
<td>EDG3904E</td>
</tr>
<tr>
<td>108</td>
<td>08</td>
<td><strong>Meaning</strong>: Your output buffer is less than 4096 bytes in size. <strong>Action</strong>: Obtain storage and set its length.</td>
<td>EDG3905E</td>
</tr>
<tr>
<td>108</td>
<td>10</td>
<td><strong>Meaning</strong>: Your output buffer is too small. The second word in your buffer contains the size you need. <strong>Action</strong>: Obtain the correct amount of storage and set its length.</td>
<td>EDG3906E</td>
</tr>
<tr>
<td>108</td>
<td>12</td>
<td><strong>Meaning</strong>: OPERATION parameter is invalid. <strong>Action</strong>: Use \texttt{OPERATION=} to specify the parameter; check your program for incorrect modifying of the parameter list.</td>
<td>EDG3907E</td>
</tr>
<tr>
<td>108</td>
<td>14</td>
<td><strong>Meaning</strong>: OUTPUT parameter is invalid. <strong>Action</strong>: Use \texttt{OUTPUT=} to specify the parameter; check your program for incorrect modifying of the parameter list.</td>
<td>EDG3908E</td>
</tr>
<tr>
<td>108</td>
<td>16</td>
<td><strong>Meaning</strong>: EXPAND parameter is invalid. <strong>Action</strong>: Use \texttt{EXPAND=} to specify the parameter; check your program for incorrect modifying of the parameter list.</td>
<td>EDG3909E</td>
</tr>
<tr>
<td>108</td>
<td>18</td>
<td><strong>Meaning</strong>: MULTI parameter is invalid. <strong>Action</strong>: Use \texttt{MULTI=} to specify the parameter; check your program for incorrect modifying of the parameter list.</td>
<td>EDG3909E</td>
</tr>
<tr>
<td>108</td>
<td>56</td>
<td><strong>Meaning</strong>: The token is already in use. <strong>Action</strong>: Use \texttt{TOKEN=} to specify a token that is not in use.</td>
<td>EDG3910E</td>
</tr>
<tr>
<td>108</td>
<td>58</td>
<td><strong>Meaning</strong>: OUTPUT=FIELDS is not supported for the subcommand specified by SUBCMDADDR=\texttt{subcmdaddr}. <strong>Action</strong>: Use \texttt{OUTPUT=}LINES or specify a different subcommand.</td>
<td>EDG3911E</td>
</tr>
<tr>
<td>108</td>
<td>60</td>
<td><strong>Meaning</strong>: The length of the subcommand specified by SUBCMDADDR=\texttt{subcmdaddr} is too large. <strong>Action</strong>: Use a smaller subcommand.</td>
<td>EDG3912E</td>
</tr>
</tbody>
</table>
### Table 4. Return and reason codes for the EDGXI 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>—</td>
<td><strong>Meaning</strong>: Environmental error. A limit, such as a storage limit, was exceeded. The operation you requested was not successfully completed. <strong>Action</strong>: Refer to the action provided with the specific reason code.</td>
<td></td>
</tr>
<tr>
<td>112 02</td>
<td></td>
<td><strong>Meaning</strong>: Unable to obtain sufficient work area storage. <strong>Action</strong>: 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><strong>Meaning</strong>: System error. An error caused by the system, rather than your program, has been encountered. The operation you requested was not successfully completed. <strong>Action</strong>: Refer to the action provided with the specific reason code.</td>
<td></td>
</tr>
<tr>
<td>116 02</td>
<td></td>
<td><strong>Meaning</strong>: DFSMSrmm is not installed. <strong>Action</strong>: Ensure DFSMSrmm is installed and active before running your program.</td>
<td>EDG3914E</td>
</tr>
<tr>
<td>116 04</td>
<td></td>
<td><strong>Meaning</strong>: 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. <strong>Action</strong>: Retry the subcommand after the cause of the error has been corrected or removed.</td>
<td>EDG3915E</td>
</tr>
<tr>
<td>116 06</td>
<td></td>
<td><strong>Meaning</strong>: An abnormal end has occurred. <strong>Action</strong>: Remove the cause of the abnormal end. Rerun your program.</td>
<td>EDG3916E</td>
</tr>
<tr>
<td>120</td>
<td>02</td>
<td><strong>Meaning</strong>: Program error has occurred while you were using the high-level API. <strong>Action</strong>: Refer to the action provided with the specific reason code.</td>
<td>EDG3918E</td>
</tr>
<tr>
<td>120 04</td>
<td></td>
<td><strong>Meaning</strong>: The LOAD for program EDGXAPI failed. <strong>Action</strong>: 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 here 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.

**Note:** SAMPLIB member EDGAPISR provides a similar example of using EDGXCI.

Macro continuation characters must be entered in column 72.

```assembly
YOURPGM CSECT
R0 EQU 0
R1 EQU 1
R2 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
*************************************************************
* Load the API module **
*************************************************************
  LOAD EP=EDGXAPI
  ST R0,APIMOD@ Save API module address
* . .
* . .
  XC MYTOKEN,MYTOKEN Ensure no token yet
  LA R4,LISTV@ List volume subcmd address
  BAL R9,BEGINCMD Begin the command
* . .
*************************************************************
* Going to reuse the resources, instead of releasing**
* resources obtained by the API for the 1st BEGIN **
*************************************************************
  LA R4,SEARCHD@ Search subcmd address
  BAL R9,BEGINCMD Begin the command
* . .
  BAL R9,MOREDATA Get more data for search
* . .
  BAL R9,RELEASE All done, release resources
* . .
*************************************************************
* Delete the API module **
*************************************************************
  DELETE EP=EDGXAPI
* . .
*************************************************************
** Call API to begin a new subcommand **
*************************************************************
BEGINCMD DS 0H
CALL EDGXCI MF=(E,MYPL),PLISTVER=MAX
  APIADDR=APIMOD@,OPERATION=BEGIN,
  TOKEN=MYTOKEN,
  SUBCMDADDR=(R4),OUTBUFADDR=APIOUTB@
  BR R9 Return
*************************************************************
** Call API to get more data for current subcommand **
*************************************************************
MOREDATA DS 0H
```
CALL2 EDGXCI MF=(E,MYPL,NOCHECK),PLISTVER=MAX, X
  OPERATION=CONTINUE,TOKEN=MYTOKEN
BR R9 Return
*******************************************************************************
** Call API to release resource such as storage and **
** loaded modules. **
*******************************************************************************
RELEASE DS 0H
REL1 EDGXCI MF=(E,MYPL,NOCHECK),PLISTVER=MAX, X
  OPERATION=RELEASE,TOKEN=MYTOKEN
BR R9 Return
*******************************************************************************
** SEARCH DATA SET SUBCOMMAND **
*******************************************************************************
SEARCHD DS 0C
  DC AL2(SEARCHDL)
  DC C'SEARCHDATASET ....'
SEARCHD 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 **
*******************************************************************************
SFDEFS DSECT
  EDGXSF
  EDGXSF
END
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 22.

To create your own program as shown in [Figure 2 on page 16](#), 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 on page 17 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.

Here is sample code that you can modify to use the high-level application programming interface.

```c++
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include "EDGXHCLU"

FILE* sfiFp;
```

Figure 3. Sample JCL for requesting LISTVOLUME information

Chapter 2. Using the object-oriented DFSMSrmm application programming interface using C++ 17
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;
    }
}

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
","DD:XMLFILE");
        exit(0);
    }

    if ( (sfiFp = fopen("DD:SFIFILE","wb,type=record")) == NULL )
    {
        printf("could not open %s
","DD:SFIFILE");
        exit(0);
    }

    pApi = new RmmApi();
    printf(" \nAPI object created \n");

    if ( pApi->openApi() == 0 )
    {
        printf("API Return Code : %d\n",pApi->getApiRC());
        printf("API Reason Code : %d\n",pApi->getApiRS());
        printf("API Message : %s\n",pApi->getMessageText());
    }
    else
    {
        printf("Could not open API \n");
exit(0);

/***************************************************************
* create RmmCommand object
***************************************************************************/

pCom = new RmmCommand(pApi);

/***************************************************************
* processes a TSO command
***************************************************************************/

rc = pCom->issueCmd(tsoCommand);

switch (rc)
{
    case 0:
        printf("Return Code : %d\n",pCom->getApiRC());
        printf("Reason Code : %d\n",pCom->getApiRS());
        printf("Message : %s\n",pCom->getMessageText());
        printSFItoFile((RmmInterface::t_outp*) pCom->getBufferSfi());
        fprintf(xmlFp,"%s\n",pCom->getBufferXml());
        break;
    case 1:
        printf("Return Code : %d\n",pCom->getApiRC());
        printf("Reason Code : %d\n",pCom->getApiRS());
        printf("Message : %s\n",pCom->getMessageText());
        printSFItoFile((RmmInterface::t_outp*) pCom->getBufferSfi());
        fprintf(xmlFp,"%s\n",pCom->getBufferXml());
        while( (pCom->getApiRC()==0) && (pCom->getApiRS()==4) )
        {
            rc = pCom->getNextEntry();
            printf("Return Code : %d\n",pCom->getApiRC());
            printf("Reason Code : %d\n",pCom->getApiRS());
            printf("Message : %s\n",pCom->getMessageText());
            printSFItoFile((RmmInterface::t_outp*) pCom->getBufferSfi());
            fprintf(xmlFp,"%s\n",pCom->getBufferXml());
        }
        break;
    case -1:
        printf("Return Code : %d\n",pCom->getApiRC());
        printf("Reason Code : %d\n",pCom->getApiRS());
        printf("Message : %s\n",pCom->getMessageText());
        break;
    default:
        printf("Return Code : %d\n",pCom->getApiRC());
        printf("Reason Code : %d\n",pCom->getApiRS());
        printf("Message : %s\n",pCom->getMessageText());
        break;
}

/***************************************************************
* destruction
***************************************************************************/

delete pCom;
delete pApi;

fclose(sfiFp);
fclose(xmlFp);
exit(0);
} /* end main */
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>
<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>
<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>
<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>RmmCommand.getBufferXml()</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>RmmCommand.getNextEntry()</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>RmmInterface.getMessageText()</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>RmmInterface.getApiRc()</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="#">EDGXCI return and reason codes on page 9</a> for information about message processing.</td>
</tr>
<tr>
<td>RmmInterface.getApiRs()</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="#">EDGXCI return and reason codes on page 9</a> for information about message processing.</td>
</tr>
<tr>
<td>RmmTransaction.runCommandXml()</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>RmmJApi.runCommandXml()</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. DFSMSrmm API Java methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| RmmJApi.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. |

### 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 4 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 4. C++ code example for writing XML output to a file

Figure 5 on page 23 shows the content of the file XMLFILE.
Most of the DFSMSrmm-produced XML tags use the SFI names described in Table 16 on page 81. 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 structured field introducers `<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:

```xml
&amp;
&amp;lt; &amp;gt;
&amp;quot;
&amp;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](http://www.ibm.com/zseries/software/xml) or from the IBM Software Delivery for System Modification Program Extended (SMP/E) installation.
Chapter 3. Using the DFSMSrmm application programming interface with web services

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

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

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

You can write Java applications that run on any platform that can use the DFSMSrmm API classes to obtain information about DFSMSrmm resources. You use the same DFSMSrmm subcommand strings that you can use with the EDGXCI application programming interface. You get output in Extensible Markup Language (XML). If you receive the output from the DFSMSrmm application programming interface as XML output, you can use an XML parser to process the returned data, or you can package the XML in order to use it as the base for displaying information for the end user. See “Receiving extensible markup language (XML) output data in the XML output buffer” on page 22 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 rmmapei.ear and is located in your file system directory /usr/lpp/dfsms/rmm/. This EAR file contains all the elements needed to implement and use the Web service. To install the DFSMSrmm Web service, use
the WebSphere Install Application. You can use either the graphical user interface or the command line tool for the installment and customization of your WebSphere environment. To develop a client application that uses the DFSMSrmm Web service, either import the EAR file into your project using Rational Application Developer (RAD) and use the definitions and codes it contains for your application, or use the sample client application shipped with DFSMSrmm, rmmSampleWSEClient.java. After your application is written, modify the installation or environment-dependent information in the EAR file so you can implement the Web service in your environment. For more information, see the general web service help file rmmwebs.txt.

The web service to be used under Apache Tomcat is shipped as a Web ARchive (WAR), called rmmapi.tc.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 20 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 /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 20). 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 rmmSampleWSEClient.java for a coding example.

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

---

### Sample Java web service client

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

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

#### Usage:

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

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

command = A valid DFSMSrmm TSO subcommand, for example, `LISTCONTROL OPTION`

A sample Java web service client, EDGSJWS1, is provided in `/usr/lpp/dfsms/rmm/rmmSampleWSClient.java`. For information on how to use the DFSMSrmm Web service sample client, see the [z/OS DFSMSrmm Implementation and Customization Guide](#).
Using persistence and parallel processing

The Web service uses a stateless session bean and enables a single command to be run and the output returned in a single request. The method
RmmJApi.runCommandXml enables a command to be run by a single method call. See Table 8 on page 21 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 6.

Do not specify it as an RMM command with multiple input lines, as shown in Figure 7.

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 then search criteria during search processing.</td>
</tr>
</tbody>
</table>

See "Controlling output from list and search type requests" on page 73 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 structured field introducers. 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 32</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 34</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 lists, single token area&quot; on page 36</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 lists, multiple token areas” on page 37 |

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

**Coding a single parameter list, single token area**

Figure 8 on page 33 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 9 on page 35 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.
Figure 9 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 9, 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 9 on page 35 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 10 for an example of this method.

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

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

Figure 10. 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 lists, single token area**

Figure 11 on page 37 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 11 on page 37 prevents you from continuing the first subcommand after you begin the second subcommand.
Coding multiple parameter lists, multiple token areas

This variation lends itself to processing in re-entrant code where subroutines can be created for commonly used code. Here is an example that 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

```
** Start the first subcommand
******************************************
XC TOKENA,TOKENA No resources/token yet
LA R4,SUBCMD1 Point to 1st subcommand
EDGXCI MF=(E-BEGINPL),PLISTVER=MAX,
  APIADDR=APIMOD0,OPERATION=BEGIN,
  TOKEN=TOKENA,
  SUBCMDADDR=(R4),OUTBUFADDR=(R3)
...  
******************************************
** Continue the subcommand
******************************************
EDGXCI MF=(E,CONTPL),PLISTVER=MAX,
  APIADDR=APIMOD0,OPERATION=CONTINUE,
  TOKEN=TOKENA,
  OUTBUFADDR=(R3)
...  
******************************************
** Done with the subcommand, release
******************************************
EDGXCI MF=(E,RELPL),PLISTVER=MAX,
  APIADDR=APIMOD0,OPERATION=RELEASE,
  TOKEN=TOKENA
...  
******************************************
** Start the second subcommand
******************************************
LA R4,SUBCMD2 Point to 2nd subcommand
EDGXCI MF=(E-BEGINPL),PLISTVER=MAX,
  APIADDR=APIMOD0,OPERATION=BEGIN,
  TOKEN=TOKENA,
  SUBCMDADDR=(R4),OUTBUFADDR=(R3)
...  
******************************************
** Continue the subcommand
******************************************
EDGXCI MF=(E,CONTPL),PLISTVER=MAX,
  APIADDR=APIMOD0,OPERATION=CONTINUE,
  TOKEN=TOKENA,
  OUTBUFADDR=(R3)
...  
******************************************
** Done with the subcommand, release
******************************************
EDGXCI MF=(E,RELPL),PLISTVER=MAX,
  APIADDR=APIMOD0,OPERATION=RELEASE,
  TOKEN=TOKENA
```

Figure 11. Multiple parameter lists, single token area
Specifying the option to free a resource

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

- Use the OPERATION=RELEASE and TOKEN=token parameters to free all resources associated with the specified token as shown in Figure 12 on page 39.
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 13.

**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 13 the OPERATION=ENDALL parameter is specified and TOKEN is not required.
Chapter 5. Using an alternative interface to the DFSMSrmm application programming interface

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

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

EDGXHINT is shipped as a load module in LINKLIB.

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

Related reading:
1. z/OS XL C/C++ User's Guide
2. 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 14.

```plaintext
//BINDPGM JOB (4378), 'BIND A PROGRAM', MSGCLASS=H, MSGLEVEL=(1,1),
// TIME=3, CLASS=A, REGION=0M, NOTIFY=&SYSUID
//*
//******************************************************************************
//*
//** BIND A C/C++ PROGRAM TO USE THE EDGXHINT INTERFACE TO RMM **
//*
//**
//** SYSLMOD: OUTPUT DATASET (HLQ.CPP.LINKLIB) MUST BE PDSE FORMAT **
//*
//******************************************************************************

//BIND EXEC PGM=IEWL, REGION=4M,
// PARM='AMODE=31, MAP, RENT'
//SYSLIB DD DSN=CCE.SCEELKEX, DISP=SHR
// DD DSN=CCE.SCEELKED, DISP=SHR
// DD DSN=CCE.SCEECP, DISP=SHR
//SYSMOD 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 14. 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 29.

### 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><strong>Function code</strong></td>
<td></td>
<td>User program</td>
</tr>
<tr>
<td></td>
<td>1. Open API (start communication)</td>
<td></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><strong>Pointer to the command buffer</strong></td>
<td>The user program needs to obtain the storage for a buffer big enough to hold the TSO subcommand to be issued. Maximum is 255 byte. EDGXHINT will read the TSO command from this buffer.</td>
<td>User program</td>
</tr>
<tr>
<td><strong>Pointer to the output buffer</strong></td>
<td>The user program needs to obtain the storage for an output buffer. Minimum recommended is 80KB. EDGXHINT will use this buffer to return the data requested.</td>
<td>User program</td>
</tr>
<tr>
<td><strong>Pointer to first message buffer</strong></td>
<td>The user program must obtain the storage for a 256 byte buffer. This buffer should always be cleared before EDGXHINT is called, to delete pre-existing content. EDGXHINT will use this buffer to return a message resulting from the last issued command, if appropriate.</td>
<td>User program</td>
</tr>
<tr>
<td><strong>Pointer to second message buffer</strong></td>
<td>The user program must obtain the storage for a 256 byte buffer. This buffer should always be cleared before EDGXHINT is called, to delete pre-existing content. EDGXHINT will use this buffer to return a second message resulting from the last issued command, if appropriate.</td>
<td>User program</td>
</tr>
<tr>
<td><strong>Message count</strong></td>
<td>Number of messages returned by EDGXHINT</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td><strong>API address</strong></td>
<td>Address of EDGXAPI. Set by OPEN function. Can be used to determine if the API is open. If not NULL, then API is open.</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td><strong>MTAB address</strong></td>
<td>Address of the DFSMSrmm message table. Set by OPEN function, used by EDGXHINT internally.</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td><strong>CMMSG address</strong></td>
<td>Address of the DFSMSrmm message routine. Set by OPEN function, used by EDGXHINT internally.</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td><strong>Token</strong></td>
<td>Token used by macro EDGXCI to identify the request. The token is created at BEGIN processing (function 3) and used by CONTINUE processing (function 4). The token is cleared (set to zero) by EDGXHINT after RELEASE processing (function 5).</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td><strong>Return code</strong></td>
<td>API return code</td>
<td>EDGXHINT</td>
</tr>
<tr>
<td><strong>Reason code</strong></td>
<td>API reason code</td>
<td>EDGXHINT</td>
</tr>
</tbody>
</table>
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 15. 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 commandStrct;  // 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 outputHeaderStrct;  // 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 outputStrct;  // 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 15. 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 42.

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 16. 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
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 16. 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 47. 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.

```c
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:

```c
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 9.
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 (SFIs) and data.
  - A structured field introducer (SFI) is a structure that separates one line or field of output data from another. Structured field introducers 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 103 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 “Structured field introducers for output data for subcommands” on page 58 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 structured field introducer:

<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>
<tr>
<td></td>
<td><strong>Byte count</strong> Description</td>
</tr>
<tr>
<td>2</td>
<td>Two-byte length. The length includes the length of the structured field introducer (8 bytes) and the length of the data following the structured field introducer.</td>
</tr>
</tbody>
</table>
Three-byte SFI identifier (ID)
One-byte SFI type modifier
One-byte (reserved)
One-byte data-type identifier

Data
Data following the structured field introducer, which can contain actual data, no data, binary zeros, or blank data.

See Appendix A, “Structured field introducers (SFIs),” on page 77, for descriptions of the structured field introducers 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 structured field introducer and its descriptive name.

**Requesting structured field introducer 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 structured field introducers and variable data.

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

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

In the examples that follow, assume that
A00001: RMMUSER.TSO.COMMAND1.

is only one data set on the volume

**Requesting line format**

Figure 17 on page 49 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 17 on page 49. The value for <line> is the SFI for each line and is followed by the data returned from specifying the RMM LISTDATASET subcommand.
Requesting field format

Figure 18 on page 50 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 18 on page 50.
Figure 18. Example of output using OUTPUT=FIELDS

50 z/OS V2R1.0 DFSMSrmm Application Programming Interface
Figure 18 on page 50

- Shows Begin and End group structured field introducers. In this example, <Begin DATASET Group> and <End DATASET Group>.
- Includes descriptive names used to identify structured field introducers. The SFI identifies the data type; and the long character <...> strings do not represent the actual size of the structured field introducers, which are only 8 bytes in length.
- Can appear to have no data. This is because structured fields can
  - Have no data (SFI only, as in this example), binary zeros, or blank characters.
  - Be omitted if they have no data.
- Shows that structured fields can be order independent. For example, VOL occurs before OWN for LISTDATASET (as shown in “LISTDATASET structured field introducers” on page 65) while OWN occurs before VOL for LISTPRODUCT (as shown in “LISTPRODUCT structured field introducers” on page 67).
- 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 109 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 5.

The examples shown in “Requesting standard output” and “Requesting expanded output” on page 52:

- 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 19.

<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(32) >x'FFFF9D90'
<CDTJ - Create Date : 4, packed decimal >x'2007339F'
<CTM - Create Time : 4, packed decimal >x'0116362F'
<FILE - Physical File Sequence : 4, bin(32) >x'00000001'
<RTDJ - Retention Date : 4, packed decimal >x'2010345F'
<XDTJ - Expiration Date : 4, packed decimal >x'2010344F'
<End DATASET Group>

Figure 19. Example of search type of output using EXPAND=NO
Refer to Appendix D, “Hexadecimal example of an output buffer,” on page 109 for a hexadecimal representation and discussion of the contents of the output buffer shown in Figure 19 on page 51.

**Requesting expanded output**

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

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

When you specify OUTPUT=FIELDS and EXPAND=YES, your SEARCHDATASET VOLUME(VOL001) subcommand might produce the output that is shown in Figure 20 on page 53.
Figure 20. Example of search type of output using OUTPUT=FIELDS, EXPAND=YES

Chapter 6. Processing the output data in the output buffer 53
Accessing return and reason codes

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

- Return codes and reason codes issued as a result of processing of your subcommand request. Refer to z/OS DFSMSrmm Managing and Using Removable Media for information about these codes.
- Return codes and reason codes associated with the API itself. These are the return codes and reason codes listed in “EDGXCI return and reason codes” on page 9 for macro EDGXCI.
- Return and reason codes from system services. DFSMSrmm uses various system services, such as catalog services, to process the subcommands from your application program. When DFSMSrmm receives a non-zero return code from a system service, the DFSMSrmm API places the return code and associated reason code in your output buffer as structured fields, along with a name to identify the service. See “System return and reason code structured field introducers” on page 57 for more information.

Accessing messages and message variables

The DFSMSrmm API can return messages and message variables in your output buffer. Figure 21 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.

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

Figure 21. Message and message variable structured fields. Message and Message Variable Structured Fields

Refer to “Messages and message variables structured field introducers” on page 57 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 98cccccc as CYCL/cccccc 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 ADDDATE or CHGTVOLUME, you can specify the TZ operand to indicate to the DFSMSrmm system the time zone offset the application is using.

DFSMSrmm converts dates and times to UTC/GMT/local time in order to store them in the DFSMSrmm control data set. Refer to the 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 structured field introducers in your output buffer:

- Structured field introducers that begin and end a resource group as described in “Begin and End Resource groups” on page 56.
- Structured field introducers that introduce a single line of output data, as described in:
  - “System return and reason code structured field introducers” on page 57
  - “Messages and message variables structured field introducers” on page 57
  - “ADD-Type of subcommands” on page 59
  - “CHANGE-Type of subcommands” on page 59
  - “DELETE-Type of subcommands” on page 60
  - “GETVOLUME subcommand” on page 60
  - “LIST-Type of subcommands” on page 60
  - “SEARCH-Type of subcommands” on page 70

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 (SFIs),” on page 77 describes all the structured fields that the DFSMSrmm API can return to your application program.

Appendix B, “Structured field introducers by subcommand,” on page 99 shows all of the structured field introducers by subcommand.
The DFSMSrmm API does not return information for all subcommands. For example, the DFSMSrmm API does not produce structured fields for a successful ADDBIN subcommand request.

Begin and End Resource groups

In the previous examples, you saw that output structured fields were grouped by a pair of unique structured field introducers as shown in Figure 22.

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

**Figure 22. Begin and End Resource group SFI sequence**

The Begin and End Resource group structured field introducers identify when output for a particular resource, such as a data set, begins and ends. The pairs of Begin and End Resource group structured field introducers are shown in Figure 23.

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

**Figure 23. Begin and End Resource group SFI pairs**

In addition to identifying the beginning and ending of output for a particular resource, the Begin and End Resource group structured field introducers shown in Figure 24 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.

```xml
<Begin ACCESS group>  <End ACCESS group>
<Begin ACTIONS group>  <End ACTIONS group>
<Begin CNTL group>  <End CNTL group>
<Begin LOCDEF group>  <End LOCDEF group>
<Begin MEDINF group>  <End MEDINF group>
<Begin MNTMSG group>  <End MNTMSG group>
<Begin MOVES group>  <End MOVES group>
<Begin OPENRULE group>  <End OPENRULE group>
<Begin OPTION group>  <End OPTION group>
<Begin PRTITION group>  <End PRTITION group>
<Begin PRODVOL group>  <End PRODVOL group>
<Begin REJECT group>  <End REJECT group>
<Begin SECCLS group>  <End SECCLS group>
<Begin STAT group>  <End STAT group>
<Begin STATUS group>  <End STATUS group>
<Begin STORE group>  <End STORE group>
<Begin TASKS group>  <End TASKS group>
<Begin VLPOOL group>  <End VLPOOL group>
<Begin VOL group>  <End VOL group>
```

**Figure 24. Begin and End Resource group SFI pairs for subgroups**

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

56  z/OS V2R1.0 DFSMSrmm Application Programming Interface
System return and reason code structured field introducers

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 25. DFSMSrmm issues return code 116 and reason code 06 when an error like this occurs.

```
<Begin SYSRETC group>
  <SVCN - service name : 16 , character: >
  <RTNC - return code : 4 , bin(32): >
  <RSNC - reason code : 4 , bin(32): >
<End SYSRETC group>
```

Figure 25. System return and reason codes

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

Messages and message variables structured field introducers

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 26.

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

Figure 26. Structured field introducers 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 27.

```
<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 27. 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 58 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 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 structured field introducers

<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 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 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>KEYT TYPF TYPT</td>
<td>SD SV</td>
<td>4 12</td>
</tr>
</tbody>
</table>

For a detailed explanation of these messages, see z/OS MVS System Messages, Vol 5 (EDG-GFS). For DFSMSrmm return and reason codes, see z/OS DFSMSrmm Managing and Using Removable Media.

Structured field introducers 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 28 on page 59. DFSMSrmm places system return codes and reason codes in your output buffer as described in System return and...
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 29.
- An error occurs for specific return and reason code combinations described in "Messages and message variables structured field introducers" on page 57 and "Structured field introducers for return and reason codes" on page 79.

**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 30 on page 60.
- When an error occurs for specific return and reason code combinations described in "Messages and message variables structured field introducers" on page 57 and "Structured field introducers for return and reason codes" on page 79.
DELETE-Type of subcommands

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

The DFSMSrmm API returns information when an error occurs for specific return and reason code combinations described in "Messages and message variables structured field introducers" on page 57 and "Structured field introducers for return and reason codes" on page 79.

GETVOLUME subcommand

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

The DFSMSrmm API returns information when:

• The GETVOLUME request was successful. The DFSMSrmm API returns volume information and owner information as shown in Figure 31.

• When an error occurs, and then only for specific return and reason code combinations described in "Messages and message variables structured field introducers" on page 57 and "Structured field introducers for return and reason codes" on page 79.

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:

• "LISTBIN structured field introducers" on page 61
• "LISTCONTROL structured field introducers" on page 61
• "LISTDATASET structured field introducers" on page 65
LISTBIN structured field introducers

The structured field introducers produced for the LISTBIN subcommand with OUTPUT=FIELDS are:

```
<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 >
<MEGN - 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(32) >
<LCDJ - Last Change Date : 4, packed decimal >
<LCTM - Last Change Time : 4, packed decimal >
<LCID - Last Change User ID : 8, character >
<LCSI - Last Change System ID : 8, character >
<LCUD - Last "User" Change Date : 4, packed decimal >
<LCUT - Last "User" Change Time : 4, packed decimal >
<End RACK/BIN Group>
```

LISTCONTROL structured field introducers

The structured field introducers produced for the LISTCONTROL subcommand with OUTPUT=FIELDS differ depending on whether the STATUS operand is specified.

The structured field introducers produced for the LISTCONTROL subcommand with OUTPUT=FIELDS are:

```
<Begin CONTROL Group>
<Begin CNTL Group>
<TZ - Time Zone : 4, bin(31) >
<MTP - CDS type : 1, bin(8) >
<MDTJ - CDS Create Date : 4, packed decimal >
<MTM - CDS Create Time : 4, packed decimal >
<UDTJ - CDS Last update date : 4, packed decimal >
<UTM - CDS Last update time : 4, packed decimal >
<RJRNF - JOURNALFULL Parmlib Value: 2, bin(15) >
<RJNS - Journal status : 1, bin(8) >
<BDTJ - Last CDS Backup Date : 4, packed decimal >
<BTM - Last CDS Backup Time : 4, packed decimal >
<JBMT - Last journal backup date : 4, packed decimal >
<JBMN - Last journal backup time : 4, packed decimal >
<XDTJ - Expiration Date : 4, packed decimal >
<XTM - Last Inven Mgt Expir Time : 4, packed decimal >
<RTM - Last CDS Extract Time : 4, packed decimal >
<DDTJ - Delete/Store Date : 4, packed decimal >
<DTM - Last Store update run time: 4, packed decimal >
<SOSJ - Last XPROC Start Date : 4, packed decimal >
<SOST - Last XPROC Start Time : 4, packed decimal >
<VDTJ - Last Inven Mgt Proc Date : 4, packed decimal >
<VTM - Last Inven Mgt VRS Time : 4, packed decimal >
<LRK - # Library Rack Numbers : 4, bin(31) >
<FRK - Free Rack Num in Library : 4, bin(31) >
```
<LBN - Bin Numbers in LOCAL : 4, bin(31) >
<FLB - Free Bin Numbers in LOCAL : 4, bin(31) >
<DBN - Bin Numbers in DISTANT : 4, bin(31) >
<FDB - Free Bins in DISTANT Loc : 4, bin(31) >
<RBN - # Bin Numbers in REMOTE : 4, bin(31) >
<FRB - Free Bin Numbers in REMOT : 4, bin(31) >
<CACT - Control Active Functions : 1, bit(8) >
<CSDT - Catalog Synchronize date : 4, packed decimal >
<CSTM - Catalog Synchronize time : 4, packed decimal >
<FCSP - Catalog Synch in progress : 1, bin(8) >
<CSVE - Stacked volume enabled : 1, bin(8) >
<X100 - EDGUX100 exit status : 1, bin(8) >
<X200 - EDGUX200 exit status : 1, bin(8) >
<X300 - EDGUX300 exit status : 1, bin(8) >
<EBIN - Extended Bin Status : 1, bin(8) >
<CDSU - CDS percentage used : 2, bin(15) >
<CSHN - Client/Server host name : 63, character >
<CSIP - Client/Server IP address : 15, character >
<UTC - Common time status enable : 1, bin(8) >
<CDSQ - CDS id ENQ name enabled : 1, bin(8) >
<RMID - RMM started procedure nam: 17, character >

<End CNTL Group>

<Begin OPTION Group>

<OPM - Operating Mode : 1, bin(8) >
<DRP - Default Retention Period : 4, bin(31) >
<MRP - Maximum Retention Period : 4, bin(31) >
<CRP - CATRETPD Retention Period: 4, bin(31) >
<MDS - CDS Data Set Name : 44, character >
<JDS - Journal Name : 44, character >
<JRNF - JOURNALFULL Parmlib Value: 2, bin(15) >
<CATS - CATSYSID value : 1, bin(8) >
<SOSP - Scratch Procedure Name : 8, character >
<BKPP - Backup Procedure Name : 8, character >
<IPL - Data Check Required in IP: 1, bin(8) >
<DTE - Installation Date Format : 1, bin(8) >
<RCF - Installation RACF Support: 1, bin(8) >
<AUD - SMF Audit Record Number : 2, bin(15) >
<SSM - SMF Security Record Number: 2, bin(15) >
<CDS - Control Data Set ID : 8, character >
<SLM - MAXHOLD Value : 2, bin(15) >
<LCT - Default Lines per Page : 2, bin(15) >
<SID - SMF System ID : 8, character >
<BLP - BLP Option : 1, bin(8) >
<NOT - Notify : 1, bin(8) >
<UNC - Uncatalog Option : 1, bin(8) >
<VRJ - VRS Job Name : 1, bin(8) >
<MSGF - Case of Message Text : 1, bin(8) >
<MOP - Master Overwrite : 1, bin(8) >
<ACCT - Accounting Source : 1, bin(8) >
<VCHG - VRSMCHANGE Value : 1, bin(8) >
<VRS - VRSEL Value : 1, bin(8) >
<PSFX - Parmlib Member Suffix : 2, character >
<PSF2 - Parmlib Member Suffix 2 : 2, character >
<VACT - VRSMIN action : 1, bin(8) >
<VMIN - VRSMIN Count Value : 4, bin(31) >
<JRNNT - Journal transaction : 1, bin(8) >
<VDRA - VRS Drop Action : 1, bin(8) >
<VDRC - VRS Drop Count : 4, bin(31) >
<VDRP - VRS Drop Percentage : 2, bin(15) >
<VREA - VRS Retain Action : 1, bin(8) >
<VREC - VRS Retain Count : 4, bin(31) >
<VREC - VRS Retain Percentage : 2, bin(15) >
<XDRA - EXPDT Drop Action : 1, bin(8) >
<XDRC - EXPDT Drop Count : 4, bin(31) >
<XDRP - EXPDT Drop Percentage : 2, bin(15) >
<DSPD - Disposition DD name : 8, character >
<DSPM - Disposition message prefi: 8, character >
Chapter 6. Processing the output data in the output buffer

<RTBY - Retain by  : 1, bin(8) >
<MVBY - Move by  : 1, bin(8) >
<GDGC - GDG cycleby : 1, bin(8) >
<GDGD - GDG duplicate : 1, bin(8) >
<PDA - PDA state  : 1, bin(8) >
<PDAC - PDA block count : 1, bin(8) >
<PDAS - PDA block size : 1, bin(8) >
<PDAL - PDA log state : 1, bin(8) >
<TVXP - Extradays retention : 1, bin(8) >
<TVXD - Extradays for TVEXTPURGE : 4, bin(31) >
<SMP - System managed tape purge: 1, bin(8) >
<SMU - System managed tape update: 1, bit(8) >
<ACS - SMS ACS support : 1, bin(8) >
<PACS - Pre-ACS support : 1, bin(8) >
<RUB - Reuse Bin at  : 1, bin(8) >
<CMD - Command Auth DSN : 1, bin(8) >
<CMDO - Command Auth Owner  : 1, bin(8) >
<MEDN - Media Name  : 8, character >
<LCTK - Local Task  : 4, bin(31) >
<SSTY - Subsystem type  : 1, bin(8) >
<SRHN - Server host name : 63, character >
<SRIP - Server IP Address  : 15, character >
<SRPN - Server port number : 4, bin(31) >
<SRTK - Server task : 4, bin(31) >
<RM - Retention Method : 1, bin(8) >
<LRED - LASTREF extra days : 4, bin(31) >
<EXRB - EXPDT RetainBy : 1, bin(8) >
<MCAT - Management class attribute  : 1, bin(8) >

<Begin OPTION Group>
<Begin SECCLS Group>
<SEC - Security Class Number  : 1, bin(8) >
<NME - Security Class Name  : 8, character >
<SCST - Security Class Status : 1, bit(8) >
<CLS - Security Class Description : 32, character >
<End SECCLS Group>
<Begin VLPOOL Group>
<PID - Pool Prefix  : 6, character >
<PSN - Pool Definition System ID : 8, character >
<PRF - Pool Def RACF Option : 1, bin(8) >
<PTP - Pool Def Pool Type : 1, bin(8) >
<XDC - Expiration Date Check  : 1, bin(8) >
<ACT - Action on Release  : 1, bit(8) >
<SCRM - Scratch mode : 1, bin(8) >
<PLN - Pool Name  : 8, character >
<MEDN - Media Name  : 8, character >
<PDS - Pool Description : 40, character >
<MOP - Master Overwrite  : 1, bin(8) >
<End VLPOOL Group>
<Begin MNTMSG Group>
<MID - Mount Message ID : 12, character >
<SMI - Offset to Message ID : 2, bin(15) >
<OVL - Offset to Volume Serial : 2, bin(15) >
<OPL - Offset Rack Num or Pool I: 2, bin(15) >
<End MNTMSG Group>
<Begin REJECT Group>
<GRK - Generic Rack Number : 6, character >
<TAC - Reject Type : 1, bin(8) >
<End REJECT Group>
<Begin LOCDEF Group>
<LDDF - Location Definition Exist: 1, bin(8) >
<L DLC - Location Name : 8, character >
<LDMT - Location Management Type : 1, bin(8) >
<L DLT - Location Type : 1, bin(8) >
<L DPR - Location Priority : 4, bin(31) >
<L DAM - Location Automove : 1, bin(8) >
<L D MN - Location Media Name : 8, character >
<End LOCDEF Group>
The structured field introducers produced for the LISTCONTROL STATUS subcommand with OUTPUT=FIELDS are:

<Begin MEDINF Group>
  <MDNF - Media Information Name : 8, character >
  <MEDT - Media Type : 1, bin(8) >
  <MDTX - External Media Type : 8, character >
  <MEDR - Media Recording Format : 1, bin(8) >
  <MDRX - External Recording Techn. : 8, character >
  <VCAP - Volume capacity : 4, bin(31) >
  <MDRP - MEDINF Replace Policy Per : 4, bin(31) >
  <MDRT - MEDINF Replace Policy Tem : 4, bin(31) >
  <MDRW - MEDINF Replace Policy Wri : 4, bin(31) >
  <MDRA - MEDINF Replace Policy Age : 4, bin(31) >
<End MEDINF Group>

<Begin PRTRITION Group>
  <PTVL - Volume Serial Number : 6, character >
  <PTVS - Volume Range Start : 6, character >
  <PTVE - Volume Range End : 6, character >
  <PTTP - Type of Partition Entry : 1, bin(8) >
  <PTSA - SMT Action for Partition : 1, bin(8) >
  <PTNA - NOSMT Action for Partition : 1, bin(8) >
  <PTNL - Location Name : 8, character >
<End PRTRITION Group>

<Begin OPENRULE Group>
  <ORVL - Volume Serial Number : 6, character >
  <ORVS - Volume Range Start : 6, character >
  <ORVE - Volume Range End : 6, character >
  <ORTP - Type of Openrule Entry : 1, bin(8) >
  <ORIA - Input Action : 1, bin(8) >
  <ORII - Input Ignore Condition : 1, bit(8) >
  <ORIR - Input Reject Condition : 1, bit(8) >
  <OROA - Output Action : 1, bin(8) >
  <OROI - Output Ignore Condition : 1, bit(8) >
  <OROR - Output Reject Condition : 1, bit(8) >
<End OPENRULE Group>

<Begin ACTIONS Group>
  <ACT - Action on Release : 1, bit(8) >
  <AST - Action Status : 1, bit(8) >
<End ACTIONS Group>

<Begin MOVES Group>
  <MFR - Source Location Name : 8, character >
  <MST - Move Status : 1, bin(8) >
  <MTO - Target Location Name : 8, character >
  <MTY - Move Type : 1, bin(8) >
<End MOVES Group>

<End CONTROL Group>
When there is no information for a subgroup, such as MOVES, for the LISTCONTROL subcommand, the DFSMSrmm API returns all of the structured field introducers 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 29 for additional information.

LISTDATASET structured field introducers

The structured field introducers produced for the LISTDATASET subcommand with OUTPUT=FIELDS are:

<Begin DATASET Group>

<DSN - Data Set Name : 44, character >
<CJBN - Job Name : 8, character >
<VOL - Volume Serial : 6, character >
<OWN - Owner : 8, character >
<DSEQ - Data Set Sequence : 4, bin(31) >
<TZ - Time Zone : 4, bin(31) >
<DEV - Device Number : 4, character >
<FILE - Physical File Sequence : 4, bin(31) >
<CDTJ - Create Date : 4, packed decimal >
<CTM - Create Time : 4, packed decimal >
<SYS - Creating system ID : 8, character >
<BLKS - Block Size : 4, bin(31) >
<BLKC - Block Count : 4, bin(31) >
<LRLCL - Logical Record Length : 4, bin(31) >
<RCFM - Record Format : 4, character >
<DC - Data Class : 8, character >
<DLWJ - Date Last Written : 4, packed decimal >
<DLRJ - Date Last Read/Referenced: 4, packed decimal >
<STEP - Step Name : 8, character >
<DD - DD Name : 8, character >
<MC - Management Class : 8, character >
<SG - Storage Group Name : 8, character >
<SC - Storage Class : 8, character >
<VMV - VRS Management Value : 8, character >
<RTDJ - Retention Date : 4, packed decimal >
<VTYP - Primary VRS Type : 1, bin(8) >
<VJBN - Primary VRS Job Name : 8, character >
<VMME - Primary VRS Name : 44, character >
<VSCN - Primary VRS Subchain name: 8, character >
<VSCD - Primary VRS Subchain date: 4, packed decimal >
### LISTOWNER structured field introducers

The structured field introducers produced for the LISTOWNER subcommand with OUTPUT=FIELDS are:

```xml
<Begin OWNER Group>
<OWN - Owner : 8, character >
<SUR - Owner's Surname : 20, character >
<FOR - Owner's Forename : 20, character >
<DPT - Owner's Department : 40, character >
<ADL1 - Address Line 1 : 40, character >
<ADL2 - Address Line 2 : 40, character >
<ADL3 - Address Line 3 : 40, character >
<ITL - Owner's Internal Tele Num: 8, character >
<ELE - Owner's Ext Telephone Num: 20, character >
<EMU - Owner's User ID : 8, character >
<END - Owner's Node : 8, character >
<VLN - Number of Volumes : 4, bin(32) >
<EML - Owner's Email Address : 63, character >
<TZ - Time Zone : 4, bin(32) >
<LCDJ - Last Change Date : 4, packed decimal >
<LCTM - Last Change Time : 4, packed decimal >
<LCID - Last Change User ID : 8, character >
<LCSI - Last Change System ID : 8, character >
<LCUD - Last "User" Change Date : 4, packed decimal >
<LCUT - Last "User" Change Time : 4, packed decimal >
<End OWNER Group>
```
LISTPRODUCT structured field introducers

The structured field introducers produced for the LISTPRODUCT subcommand with OUTPUT=FIELDS are:

<Begin PRODUCT Group>
<PNUM - Software Product Number : 8, character >
<VER - Software Product Version : 6, character >
<OWN - Owner : 8, character >
<PNAME - Product Software Name : 30, character >
<VLN - Number of Volumes : 4, bin(32) >
<TZ - Time Zone : 4, bin(32) >
<LCDJ - Last Change Date : 4, packed decimal >
<LCTM - Last Change Time : 4, packed decimal >
<LCID - Last Change User ID : 8, character >
<LCSI - Last Change System ID : 8, character >
<LCLUD - Last "User" Change Date : 4, packed decimal >
<LCLUDT - Last "User" Change Time : 4, packed decimal >
<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>

The PRODVOL group is repeated for each product volume.

LISTTRACK structured field introducers

The structured field introducers produced for the LISTTRACK subcommand with OUTPUT=FIELDS are:

<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(32) >
<LCDJ - Last Change Date : 4, packed decimal >
<LCTM - Last Change Time : 4, packed decimal >
<LCID - Last Change User ID : 8, character >
<LCSI - Last Change System ID : 8, character >
<LCUD - Last "User" Change Date : 4, packed decimal >
<LCUT - Last "User" Change Time : 4, packed decimal >
<End RACK/BIN Group>

LISTVOLUME structured field introducers

The structured field introducers produced for the LISTVOLUME subcommand with OUTPUT=FIELDS are:

<Begin VOLUME Group>
<Begin VOL Group>
<VOL - Volume Serial : 6, character >
<RCK - Rack or Bin Number : 6, character >
<OWN - Owner : 8, character >
<TZ - Time Zone : 4, bin(31) >
<CJBN - Job Name : 8, character >
<CDTJ - Create Date : 4, packed decimal >
<CTM - Create Time : 4, packed decimal >
<ADTJ - Assigned Date : 4, packed decimal >
<ATM - Assigned Time : 4, packed decimal >
<XDTJ - Expiration Date : 4, packed decimal >
<XDSB - Expiry date set by : 1, bin(8) >
<OXDJ - Original Expiration Date : 4, packed decimal >
<End VOL Group>
<End VOLUME Group>
LISTVRS structured field introducers

The structured field introducers produced for the LISTVRS subcommand with OUTPUT=FIELDS are:

<Begin VRS Group>
<VRS - Vital Record Specification: 44, character >
<TYP - VRS Type : 1, bit(8) >
<VJBN - Primary VRS Job Name : 8, character >
<VRC - Vital Record Count : 4, bin(32) >
<RET - Retention Type : 3, bin(8) >
<VDD - VRS Delay Days : 2, bin(15) >
<LOC - Location : 8, character >
<SC1 - Store Number : 4, bin(32) >
<PRTY - Priority : 4, bin(32) >
<NVRS - Next VRS Name : 8, character >
<OWN - Owner : 8, character >
<DESC - Volume or VRS Description: 30, character >
<TZ - Time Zone : 4, bin(32) >
<DDTJ - Delete/Store Date : 4, packed decimal >
<VANX - Next VRS Type : 1, bin(8) >
<VRSI - Scratch Immediate : 1, bin(8) >
<VRAI - Expiration date ignore : 1, bin(8) >

Chapter 6. Processing the output data in the output buffer 69
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 29 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 structured field introducers

The output that DFSMSrmm returns when you specify the SEARCHBIN subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters is:

```
<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(32) >
<End RACK/BIN Group>
```

SEARCHDATASET structured field introducers

The output DFSMSrmm returns when you specify the SEARCHDATASET subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters is:

```
<Begin DATASET Group>
  <DSN - Data Set Name : 44, character >
  <VOL - Volume Serial : 6, character >
  <OWN - Owner : 8, character >
  <TZ - Time Zone : 4, bin(32) >
<End DATASET Group>
```
The expanded output that DFSMSrmm returns when you specify the SEARCHDATASET subcommand with the OUTPUT=FIELDS and EXPAND=YES parameters is the same as shown in "LISTDATASET structured field introducers" on page 65 for LISTDATASET.

SEARCHOWNER structured field introducers

The output DFSMSrmm returns when you specify the SEARCHOWNER subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters is:

<Begin OWNER group>
<OWN - owner : 8, character: >
<SUR - owner's surname : 20, character: >
<FOR - owner's forename : 20, character: >
<DPT - owner's department : 40, character: >
<ADL - address line : 40, character: >
<ADL - address line : 40, character: >
<ADL - address line : 40, character: >
<ITL - owner's internal tel num : 8, character: >
<ETL - owner's external tel num : 20, character: >
<EMU - owner's user ID : 8, character: >
<EMN - owner's node : 8, character: >
<VLN - number of volumes : 4, bin(32): >
<EML - owner's email address : 63, character: >
<TZ - time zone : 4, bin(32): >
<End OWNER group>

SEARCHPRODUCT structured field introducers

The output DFSMSrmm returns when you specify the SEARCHPRODUCT subcommand and the EDGXCI macro OUTPUT=FIELDS parameter is:

<Begin PRODUCT Group>
<PNUM - Software Product Number : 8, character >
<VER - Software Product Version : 6, character >
<OWN - Owner : 8, character >
<PNAME - Product Software Name : 30, character >
<PDSC - Product Description : 32, character >
<VLN - Number of Volumes : 4, bin(32) >
<TZ - Time Zone : 4, bin(32) >
<LCDJ - Last Change Date : 4, packed decimal >
<LCMT - Last Change Time : 4, packed decimal >
<LCID - Last Change User ID : 8, character >
<LCST - Last Change System ID : 8, character >
<LCUD - Last "User" Change Date : 4, packed decimal >
<LCUT - Last "User" Change Time : 4, packed decimal >
<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>

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.
SEARCHRACK structured field introducers

The output DFSMSrmm returns when you specify the SEARCHRACK subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters is:

```xml
<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(32) >
<End RACK/BIN Group>
```

SEARCHVOLUME structured field introducers

The output DFSMSrmm returns when you specify the SEARCHVOLUME subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters is:

```xml
<Begin VOLUME Group>
  <VOL - Volume Serial     : 6, character >
  <OWN - Owner             : 8, character >
  <RCK - Rack or Bin Number: 6, character >
  <TZ - Time Zone          : 4, bin(32) >
  <ADTJ - Assigned Date    : 4, packed decimal >
  <XDTJ - Expiration Date  : 4, packed decimal >
  <RTDJ - Retention Date   : 4, packed decimal >
  <LOC - Location          : 8, character >
  <INTR - Volume Intransit Status: 1, bin(8) >
  <HLOC - Home Location    : 8, character >
  <DSC - Data Set Count    : 4, bin(32) >
  <VST - Volume Status     : 1, bit(8) >
  <AVL - Volume Availability: 1, bit(8) >
  <LBL - Volume Label      : 1, bit(8) >
  <MEDT - Media Type       : 1, bin(8) >
  <MEDR - Media Recording Format: 1, bin(8) >
  <MEDC - Media Compaction : 1, bin(8) >
  <MEDA - Media Special Attributes: 1, bin(8) >
  <PEND - Actions Pending  : 1, bit(8) >
  <LOAN - Loan Location    : 8, character >
  <GEST - Destination Name : 8, character >
  <DSR - Data Set Recording: 1, bin(8) >
  <SEQ - Volume Sequence   : 4, bin(32) >
  <MEDN - Media Name       : 8, character >
  <LVC - Current label version : 1, bin(8) >
  <LVN - Required label version : 1, bin(8) >
<End VOLUME Group>
```

The expanded output that DFSMSrmm returns when you specify the SEARCHVOLUME subcommand with the OUTPUT=FIELDS and EXPAND=YES parameters is the same as shown in "LISTVOLUME structured field introducers" on page 67 for LISTVOLUME.

SEARCHVRS structured field introducers

The output DFSMSrmm returns when you specify the SEARCHVRS subcommand and the EDGXCI macro OUTPUT=FIELDS and EXPAND=NO parameters is:

```xml
<Begin VRS Group>
  <VRS - Vital Record Specification: 44, character >
  <TYP - VRS Type               : 1, bit(8) >
  <VJBN - Primary VRS Job Name  : 8, character >
  <RET - Retention Type         : 3, bin(8) >
  <LOC - Location               : 8, character >
<End VRS Group>
```
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 30.

- 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> structured field introducers rather than <LINE> structured field introducers.
- 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 32 on page 75, Figure 33 on page 75, and Figure 34 on page 76 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 32 on page 75
- The group begins with the <Begin RACK or BIN group>.
- The structured fields between the Begin and End RACK group structured field introducers are all introduced by a <LINE> SFL.
- 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.
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.

<table>
<thead>
<tr>
<th>Rack</th>
<th>Medianame</th>
<th>Volume</th>
<th>Status</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>020610</td>
<td>CART3480</td>
<td>020610</td>
<td>IN USE</td>
<td>SHELF</td>
</tr>
<tr>
<td>020742</td>
<td>CART3480</td>
<td>020742</td>
<td>IN USE</td>
<td>SHELF</td>
</tr>
<tr>
<td>021042</td>
<td>CART3480</td>
<td>021042</td>
<td>IN USE</td>
<td>SHELF</td>
</tr>
</tbody>
</table>

...  

<table>
<thead>
<tr>
<th>Rack</th>
<th>Medianame</th>
<th>Volume</th>
<th>Status</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>030311</td>
<td>CART3480</td>
<td>030311</td>
<td>IN USE</td>
<td>SHELF</td>
</tr>
<tr>
<td>030318</td>
<td>CART3480</td>
<td>030318</td>
<td>IN USE</td>
<td>SHELF</td>
</tr>
</tbody>
</table>

Figure 32. CONTINUE example, first output buffer

<Begin RACK or BIN group>

<table>
<thead>
<tr>
<th>Rack</th>
<th>Medianame</th>
<th>Volume</th>
<th>Status</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>031086</td>
<td>CART3480</td>
<td>031086</td>
<td>IN USE</td>
<td>SHELF</td>
</tr>
<tr>
<td>031568</td>
<td>CART3480</td>
<td>031568</td>
<td>IN USE</td>
<td>SHELF</td>
</tr>
<tr>
<td>031599</td>
<td>CART3480</td>
<td>031599</td>
<td>IN USE</td>
<td>TRON</td>
</tr>
</tbody>
</table>

...  

<table>
<thead>
<tr>
<th>Rack</th>
<th>Medianame</th>
<th>Volume</th>
<th>Status</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>032848</td>
<td>CART3480</td>
<td>032848</td>
<td>IN USE</td>
<td>SHELF</td>
</tr>
<tr>
<td>032898</td>
<td>CART3480</td>
<td>032898</td>
<td>IN USE</td>
<td>SHELF</td>
</tr>
</tbody>
</table>

<MSGL>EDG3203I SEARCH COMPLETE - MORE ENTRIES MAY EXIST  
<MSGL>EDG3012I 90 ENTRIES LISTED

<End RACK or BIN group>

Figure 33. CONTINUE example, second output buffer

In Figure 33
- There are no header lines in the second output buffer.
- There are only 16 output data lines (the LINE structured field introducers).
- 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 34:

- 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.

Figure 34. CONTINUE example, third (Last) output buffer
Appendix A. Structured field introducers (SFIs)

This section defines the structured field introducers (SFIs) used by the DFSMSrmm API to identify fields in API output.

Structured field introducer (SFI) format

All structured field introducers 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 “Compound SFI.”)</td>
</tr>
<tr>
<td></td>
<td>• 9=(4 bytes) Packed decimal Julian date: yyyydddC</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 structured field introducer 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 a structured field introducer 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 structured field introducers 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>

**Structured field introducers for Begin and End Resource groups**

Begin and End Resource group structured field introducers 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 structured field introducers that identify Begin and End resource groups.

**Table 13. Begin and End Resource 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>
<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>PRTITION within CONTROL</td>
</tr>
</tbody>
</table>
Table 13. Begin and End Resource group structured field introducers (continued)

<table>
<thead>
<tr>
<th>Begin - End IDs</th>
<th>Resource Group</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>X'03C000' - X'03C080'</td>
<td>STATUS - within CONTROL</td>
</tr>
<tr>
<td>X'03D000' - X'03D080'</td>
<td>TASKS – within CONTROL</td>
</tr>
</tbody>
</table>

Structured field introducers for return and reason codes

The structured field introducers shown in Table 14 provide return codes and reason codes in your output buffer.

The DFSMSrmm API issues the return and reason code structured field introducers only when the subcommand fails. Each return and reason code pair is grouped within the SYSRETC group. The FRC and FRS structured field introducers are used for return and reason codes that are returned from OAM. The RSNC and RTNC structured field introducers 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 structured field introducer 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 structured field introducers

<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>
<tr>
<td>X'404000'</td>
<td>SVCN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Service name</td>
<td>Any subcommand</td>
</tr>
</tbody>
</table>
Structured field introducers for messages and message variables

The structured field introducers 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 structured field introducers are always grouped within the MESSAGE group. The MSGL structured field introducers are grouped within the MESSAGE group when the DFSMSrmm API is unable to determine which subcommand type the message is for. One or more structured field introducers other than ENTN might follow MSGN as described in “Messages and message variables structured field introducers” on page 57.

Table 15. Message structured field introducers

<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 Min 0, Max 10-digit</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>

Structured field introducers for subcommand output data

The structured field introducers described in Table 16 on page 81 introduce subcommand output data in your output buffer. These structured field introducers are always grouped within a pair of Begin and End Resource group structured field introducers.

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 structured field introducers

<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 SO</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</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 80=PENDING_RELEASE 40=VITAL_RECORD 20=ON_LOAN 08=OPEN</td>
<td>LV SV</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'809310'</td>
<td>BESK</td>
<td>12</td>
<td>Binary(32)</td>
<td>CA Tape Encryption key index, 4 byte hex value</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'80A000'</td>
<td>BIN</td>
<td>14</td>
<td>Character (fixed length) 6-character alphanumeric bin number</td>
<td></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>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'80D0B0'</td>
<td>BLK6</td>
<td>16</td>
<td>Binary(64)</td>
<td>Total block count</td>
<td>LD</td>
</tr>
<tr>
<td>X'80E000'</td>
<td>BLP</td>
<td>9</td>
<td>Binary(8)</td>
<td>BLP option: 0=RMM 1=NORMM</td>
<td>LC</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>
</tbody>
</table>
Table 16. Command structured field introducers (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'811000'</td>
<td>CACT</td>
<td>9</td>
<td>Bit(8)</td>
<td>Control active functions: '80'=BACKUP, '40'=RESTORE, '20'=VERIFY, '08'=EXTRACT, '04'=DSTORE, '02'=VRSEL</td>
<td>LC</td>
</tr>
<tr>
<td>X'811800'</td>
<td>CATS</td>
<td>9</td>
<td>Binary(8)</td>
<td>CATSYSID value: 0=SET, 1=NOTSET, 2=*</td>
<td>LC</td>
</tr>
<tr>
<td>X'812000'</td>
<td>CDS</td>
<td>16</td>
<td>Character</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: 0=Disabled, 1=Enabled</td>
<td>LC</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 date format</td>
<td>Create date</td>
<td>LC</td>
</tr>
<tr>
<td>X'814000'</td>
<td>CJBNS</td>
<td>16</td>
<td>Character</td>
<td>Job name</td>
<td>LC</td>
</tr>
<tr>
<td>X'815000'</td>
<td>CLIB</td>
<td>16</td>
<td>Character</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</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: 0=No, 1=Yes</td>
<td>LC</td>
</tr>
<tr>
<td>X'8169A0'</td>
<td>CMDO</td>
<td>9</td>
<td>Binary(8)</td>
<td>Command Authorization - based on owner: 0=No, 1=Yes</td>
<td>LC</td>
</tr>
<tr>
<td>X'817000'</td>
<td>CNT</td>
<td>12</td>
<td>Binary(32)</td>
<td>Bin, rack, or volume count: Min 0, Max 5-digit</td>
<td>AB AR AV DB DR</td>
</tr>
<tr>
<td>X'817820'</td>
<td>CPGM</td>
<td>16</td>
<td>Character</td>
<td>Creating program name</td>
<td>LD SD(e)</td>
</tr>
<tr>
<td>X'817890'</td>
<td>CRAT</td>
<td>12</td>
<td>Binary(32)</td>
<td>Compression ratio in hundredths</td>
<td>LC LD LV</td>
</tr>
<tr>
<td>X'817900'</td>
<td>CRID</td>
<td>16</td>
<td>Character</td>
<td>File 1 create user ID</td>
<td>LV VOL, SV(e)</td>
</tr>
<tr>
<td>X'818000'</td>
<td>CRP</td>
<td>12</td>
<td>Binary(32)</td>
<td>CATRETPD retention period: Min 0, Max 4-digit</td>
<td>LC</td>
</tr>
<tr>
<td>X'818000'</td>
<td>CSDT</td>
<td>12</td>
<td>Packed decimal date</td>
<td>Catalog synchronize date</td>
<td>LC</td>
</tr>
<tr>
<td>X'819000'</td>
<td>CSG</td>
<td>16</td>
<td>Character</td>
<td>Current storage group name</td>
<td>AV CV</td>
</tr>
<tr>
<td>X'819200'</td>
<td>CSHN</td>
<td>71</td>
<td>Character</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>53</td>
<td>Character</td>
<td>Client IP address 1-to-45 numeric characters including colon, 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(e)</td>
</tr>
<tr>
<td>X'81A300'</td>
<td>CTNR</td>
<td>24</td>
<td>Character</td>
<td>In container</td>
<td>LV STORE</td>
</tr>
<tr>
<td>X'81A600'</td>
<td>DBIN</td>
<td>14</td>
<td>Character</td>
<td>Numeric: 0-999999 or 6 alphanumeric character destination bin number</td>
<td>LV</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'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>DEIST</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 LS SD(e) SS 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>X'824000'</td>
<td>DLTWJ</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'826000'</td>
<td>DPT</td>
<td>48</td>
<td>Character (variable length)</td>
<td>Owner's department</td>
<td>LO SO</td>
</tr>
<tr>
<td>X'827000'</td>
<td>DRF</td>
<td>12</td>
<td>Binary(32)</td>
<td>Default retention period: Min 0, Max 93000</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 LV SD(e) SV(e)</td>
</tr>
<tr>
<td>X'82A000'</td>
<td>DNS</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'82B300'</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=STORE 1=STORE_BUILTIN 2=MANUAL 3=AUTO 4=STORE_BINS 5=STORE_NOBINS</td>
<td>LV</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'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:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2=E</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3=I</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4=J</td>
<td></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:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=DISABLED</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=ENABLED</td>
<td></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'82DO00'</td>
<td>EMN</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Owner's node</td>
<td>LO SO</td>
</tr>
<tr>
<td>X'82F000'</td>
<td>EMU</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Owner's user ID</td>
<td>LO SO</td>
</tr>
<tr>
<td>X'830000'</td>
<td>ETL</td>
<td>28</td>
<td>Character (variable length)</td>
<td>Telephone number</td>
<td>LO SO</td>
</tr>
<tr>
<td>X'830800'</td>
<td>EXRB</td>
<td>9</td>
<td>Binary(8)</td>
<td>Retained By</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Volume = 0</td>
<td>OPT, LV, VOL, SV(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Firstfile = 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Set = 2</td>
<td></td>
</tr>
<tr>
<td>X'831000'</td>
<td>FCD</td>
<td>12</td>
<td>Character (variable length)</td>
<td>Feature code</td>
<td>LP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 1, Max 10-digit</td>
<td>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:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=NO</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=YES</td>
<td></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></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 6-digit</td>
<td></td>
</tr>
<tr>
<td>X'833000'</td>
<td>FILE</td>
<td>12</td>
<td>Binary(32)</td>
<td>Physical file sequence Min 1,</td>
<td>LD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Max 4-digit</td>
<td>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:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 6-digit</td>
<td></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:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 6-digit</td>
<td></td>
</tr>
<tr>
<td>X'837000'</td>
<td>FRK</td>
<td>12</td>
<td>Binary(32)</td>
<td>Free rack numbers in library:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 10-digit</td>
<td></td>
</tr>
<tr>
<td>X'837800'</td>
<td>GDGC</td>
<td>9</td>
<td>Binary(8)</td>
<td>GDG: CYCLEBY:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=Generation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=Create order</td>
<td></td>
</tr>
<tr>
<td>X'837805'</td>
<td>GDGD</td>
<td>9</td>
<td>Binary(8)</td>
<td>GDG: DUPLICATE:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=Bump from sub chain</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=Drop from chain</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2=Keep</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3=Count</td>
<td></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'838F40'</td>
<td>HLD</td>
<td>9</td>
<td>Binary(8)</td>
<td>0=Hold No</td>
<td>LV, SV(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=Hold Yes</td>
<td></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>
</tbody>
</table>
### Table 16. Command structured field introducers (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'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>IRMM</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, SO</td>
</tr>
<tr>
<td>X'83CB00'</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'83D000'</td>
<td>JDM</td>
<td>12</td>
<td>Packed decimal time format</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=Enabled, 1=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'841500'</td>
<td>LCDJ</td>
<td>12</td>
<td>Packed decimal Julian Date format</td>
<td>Last change date</td>
<td>LB LD LO LP LR LV LS SD(e) SP(e) SS(e)</td>
</tr>
<tr>
<td>X'842000'</td>
<td>LCID</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Last change user IDID starts with asterisk (*) for change made by DFSMSrmm</td>
<td>LB LD LO LP LR LS LV SD(e) SP(e) SS(e) SV(e)</td>
</tr>
<tr>
<td>X'842500'</td>
<td>LCSI</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Last change system ID</td>
<td>LB LD LO LP LR LS LV SD(e) SP(e) SS(e) 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'843500'</td>
<td>LCTM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last change time</td>
<td>LB LD LO LP LR LS LV SD(e) SP(e) SS(e) SV(e)</td>
</tr>
<tr>
<td>X'843600'</td>
<td>LCU/D</td>
<td>12</td>
<td>Packed decimal Julian Date format</td>
<td>Last user change date</td>
<td>LB LD LO LP LR LS LV SD(e) SP(e) SS(e) SV(e)</td>
</tr>
<tr>
<td>X'843700'</td>
<td>LCU/T</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last user change time</td>
<td>LB LD LO LP LR LS LV SD(e) SP(e) SS(e) SV(e)</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'843800'</td>
<td>LDD</td>
<td>16</td>
<td>Character (fixed length)</td>
<td>Last used DD name</td>
<td>LD SD(e)</td>
</tr>
</tbody>
</table>

**Appendix A. Structured field introducers (SFIs)**
### Table 16. Command structured field introducers (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'845000'</td>
<td>LDEV</td>
<td>12</td>
<td>Character</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</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:</td>
<td>LC</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=AUTO</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=STORE</td>
<td></td>
</tr>
<tr>
<td>X'848000'</td>
<td>LDMN</td>
<td>16</td>
<td>Character</td>
<td>Location media name</td>
<td>LC</td>
</tr>
<tr>
<td>X'849000'</td>
<td>LDMT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Location management type:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=UNDEFINED</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=BIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2=NOBINS</td>
<td></td>
</tr>
<tr>
<td>X'84A000'</td>
<td>LDPR</td>
<td>12</td>
<td>Binary(32)</td>
<td>Location priority:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 4-digit</td>
<td></td>
</tr>
<tr>
<td>X'84A100'</td>
<td>LDAM</td>
<td>9</td>
<td>Binary(8)</td>
<td>Automove:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0= No</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1= Yes</td>
<td></td>
</tr>
<tr>
<td>X'84B000'</td>
<td>LINE</td>
<td>264</td>
<td>Character</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</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</td>
<td>Loan location</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'84D000'</td>
<td>LOC</td>
<td>16</td>
<td>Character</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:</td>
<td>LV SV(e)</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'84E760'</td>
<td>LPGM</td>
<td>16</td>
<td>Character</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:</td>
<td>LD SD(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'84F800'</td>
<td>LRED</td>
<td>12</td>
<td>Binary(32)</td>
<td>Last reference extra days</td>
<td>LC, LDSD(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 93000</td>
<td></td>
</tr>
<tr>
<td>X'850000'</td>
<td>LRK</td>
<td>12</td>
<td>Binary(32)</td>
<td>Library rack numbers:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min 0, Max 10-digit</td>
<td></td>
</tr>
<tr>
<td>X'850370'</td>
<td>LSTP</td>
<td>16</td>
<td>Character</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:</td>
<td>LV SV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=No version specified</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=Label version 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3=Label version 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4=Label version 4</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'850A00'</td>
<td>LVN</td>
<td>9</td>
<td>Binary(8)</td>
<td>Required label version: 0=No version specified 3=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'851200'</td>
<td>MCAT</td>
<td>9</td>
<td>Binary(8)</td>
<td>SMS Management class attributes enabling 0 = NONE 1 = ALL 2 = VRSELXDI</td>
<td>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>
<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) LC</td>
</tr>
<tr>
<td>X'854000'</td>
<td>MEDA</td>
<td>9</td>
<td>Binary(8)</td>
<td>Media special attributes: 0=None 1=RDCOMPAT</td>
<td>LV SV</td>
</tr>
<tr>
<td>X'855000'</td>
<td>MEDC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Media compaction 0=UNDEFINED 1=NO 2=YES</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: 0=NON-CARTRIDGE 1=18TRK 2=36TRK 3=128TRK 4=256TRK 5=384TRK 6=E_FMT1 7=E_FMT2 8=E_FMT2 9=E_FMT3 10=E_FMT3 11=E_FMT4 12=E_FMT4</td>
<td>LV SV LC</td>
</tr>
</tbody>
</table>
### Table 16. Command structured field introducers (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'858000'</td>
<td>MEDT</td>
<td>9</td>
<td>Binary(8)</td>
<td>Media type:</td>
<td>LV SV LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=UNDEFINED</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=CST</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2=ECCST</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3=HPCT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4=EHPCT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5=ETC/MEDIA5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6=EWTC/MEDIA6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7=EETC/MEDIA7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8=EEWTC/MEDIA8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9=EXTC/MEDIA9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10=EXWTC/MEDIA10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11=EATC/MEDIA11</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12=EAWTC/MEDIA12</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13=EAEVC/MEDIA13</td>
<td></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: 0=AUTO 1=MANUAL</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: 0=ADD 1=LAST 3=MATCH 4=USER</td>
<td>LC</td>
</tr>
<tr>
<td>X'85D000'</td>
<td>MRP</td>
<td>12</td>
<td>Binary(32)</td>
<td>Maximum retention period: Min 0, Max 93000 -1 (negative) means unlimited retention.</td>
<td>LC</td>
</tr>
<tr>
<td>X'85E000'</td>
<td>MSGF</td>
<td>9</td>
<td>Binary(8)</td>
<td>Message text case: 0=MIXED 1=UPPER</td>
<td>LC</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=MANSUAL 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'868000'</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>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'869000'</td>
<td>OAC</td>
<td>9</td>
<td>Binary(8)</td>
<td>Owner access</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=READ</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=UPDATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2=ALTER</td>
<td></td>
</tr>
<tr>
<td>X'86A000'</td>
<td>OBMN</td>
<td>16</td>
<td>Character</td>
<td>Old bin number media name</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(variable length)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'86B000'</td>
<td>OBN</td>
<td>14</td>
<td>Character</td>
<td>Old bin number</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(fixed length)</td>
<td></td>
<td></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</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=NO</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=YES</td>
<td></td>
</tr>
<tr>
<td>X'86C000'</td>
<td>OLOC</td>
<td>16</td>
<td>Character</td>
<td>Old location</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(variable length)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'86C100'</td>
<td>OLON</td>
<td>16</td>
<td>Character</td>
<td>Old loan location</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(variable length)</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 Min 1, Max 3-digit</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LC Position in the message</td>
<td></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>1=M</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2=R</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3=W</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4=P</td>
<td></td>
</tr>
<tr>
<td>X'86E8A0'</td>
<td>ORIA</td>
<td>9</td>
<td>Binary(8)</td>
<td>Input action: 0=ACCEPT 1=IGNORE 2=REJECT</td>
<td>LC</td>
</tr>
<tr>
<td>X'86E8A8'</td>
<td>ORII</td>
<td>9</td>
<td>Bit(8)</td>
<td>Input IGNORE condition (BY): X'80'=SPECIFIC X'40'=NONSPECIFIC X'C0'=ANY</td>
<td>LC</td>
</tr>
<tr>
<td>X'86E8B8'</td>
<td>ORIR</td>
<td>9</td>
<td>Bit(8)</td>
<td>Input REJECT condition (BY): X'80'=SYSID X'40'=CATLG</td>
<td>LC</td>
</tr>
<tr>
<td>X'86EA00'</td>
<td>OROA</td>
<td>9</td>
<td>Binary(8)</td>
<td>Output action: 0=ACCEPT 1=IGNORE 2=REJECT</td>
<td>LC</td>
</tr>
<tr>
<td>X'86EA08'</td>
<td>OROI</td>
<td>9</td>
<td>Bit(8)</td>
<td>Output IGNORE condition (BY): X'80'=SPECIFIC X'40'=NONSPECIFIC X'C0'=ANY</td>
<td>LC</td>
</tr>
<tr>
<td>X'86EA18'</td>
<td>OROR</td>
<td>9</td>
<td>Bit(8)</td>
<td>Output REJECT condition (BY): X'80'=SYSID X'40'=CATLG</td>
<td>LC</td>
</tr>
<tr>
<td>X'86EF08'</td>
<td>ORTP</td>
<td>9</td>
<td>Binary(8)</td>
<td>Type of open rule entry: 0=RMM 1=NORMM</td>
<td>LC</td>
</tr>
<tr>
<td>X'86EF80'</td>
<td>ORVS</td>
<td>14</td>
<td>Character</td>
<td>Volume range start</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(variable length)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'86EF85'</td>
<td>ORVL</td>
<td>14</td>
<td>Character</td>
<td>Volume serial number, specific or generic</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(variable length)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'86EF88'</td>
<td>ORVE</td>
<td>14</td>
<td>Character</td>
<td>Volume range end</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(variable length)</td>
<td></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: Min 1, Max 3-digit</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LC Position in the message</td>
<td></td>
</tr>
<tr>
<td>X'86F500'</td>
<td>OVOL</td>
<td>14</td>
<td>Character</td>
<td>Old volume</td>
<td>LB SB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(fixed length)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'870000'</td>
<td>OWN</td>
<td>16</td>
<td>Character</td>
<td>Owner</td>
<td>GV LD LO LP LS LV SD(e) SO SP SS SV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(variable length)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'871000'</td>
<td>OXDJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>LD LV SD(e) SV(e)</td>
<td></td>
</tr>
</tbody>
</table>
Table 16. Command structured field introducers (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>
<th>Actions pending:</th>
</tr>
</thead>
</table>
| X'871800'    | PACS     | 9          | Binary(8)     | PREACS
0=NO
1=YES                        | LC         |                 |
| X'871E00'    | PDA      | 9          | Binary(8)     | PDA state:
0=Off
1=On
2=None                      | LC         |                 |
| X'871E10'    | PDAC     | 9          | Binary(8)     | PDA block count: Numeric
2-255                      | LC         |                 |
| X'871E30'    | PDA      | 9          | Binary(8)     | PDA log state: 0=Off=On
LC                          | LC         |                 |
| X'871E40'    | PDAS     | 9          | Binary(8)     | PDA block size: Numeric
1-31                      | LC         |                 |
| X'872000'    | PDS      | 48         | Character (variable length) | Pool description | LC |
| X'873000'    | PDSC     | 40         | Character (variable length) | Product description | LP SP(e) |
| X'874000'    | PEND     | 9          | Bit(8)        | Actions pending:
"'80'"=SCRATCH
"'40'"=REPLACE
"'20'"=INIT
"'10'"=ERASE
"'08'"=RETURN
"'04'"=NOTIFY | LV SV |
| X'875000'    | PID      | 14         | Character (variable length) | Pool prefix | LC LR SR |
| X'876000'    | PNME     | 38         | Character (variable length) | Software product name | LP SP |
| X'877000'    | PLN      | 16         | Character (variable length) | Pool name | LC |
| X'878000'    | PNUM     | 16         | Character (variable length) | Software product number | LP LV SP SV(e) |
| X'879000'    | PRD      | 12         | Binary(32)    | Permanent read errors: Min 0, Max 5-digit | LV SV(e) |
| X'87A000'    | PRF      | 9          | Binary(8)     | Pool definition RACF® (A component of the Security Server for z/OS) option: 0=NO
1=YES | LC |
| X'87B000'    | PRTY     | 12         | Binary(32)    | Priority: Min 0, Max 4-digit | LS SS |
| X'87C000'    | PSFX     | 10         | Character (fixed length) | Parmlib member suffix | LC |
| X'87C010'    | PSF2     | 10         | Character (fixed length) | Second parmlib member suffix | LC |
| X'87D000'    | PSN      | 16         | Character (variable length) | Pool definition system ID | LC |
| X'87D300'    | PSZ6     | 22         | Compound (Binary(8) Factor, Binary(64) Value) | Physical space used | LD LV |
| X'87D800'    | PTNA     | 9          | Binary(8)     | NOSMT action for partition entry: 0=ACCEPT 1=IGNORE | LC |
| X'87D900'    | PTML     | 16         | Character (variable length) | Location name | LC |
| X'87E000'    | PTP      | 9          | Binary(8)     | Pool definition pool type: 0=SCRATCH 1=RACK | LC |
| X'87E800'    | PTSA     | 9          | Binary(8)     | SMT action for partition entry: 0=ACCEPT 1=IGNORE | LC |
| X'87EB00'    | PTPP     | 9          | Binary(8)     | Type of partition entry: 0=RMM
1=NORMM | LC |
<p>| X'87EC00'    | PTVS     | 14         | Character (variable length) | Volume range start | LC |
| X'87EC08'    | PTVL     | 14         | Character (variable length) | Volume serial number, specific or generic | LC |
| X'87EC0F'    | PTVE     | 14         | Character (variable length) | Volume range end | LC |
| X'87FD00'    | PVL      | 14         | Character (fixed length) | Previous volume: 1 - 6 character | LV SV(e) |
| X'880000'    | PWT      | 12         | Binary(32)    | Permanent write errors: Min 0, Max 5-digit | LV SV(e) |
| X'881000'    | RBN      | 12         | Binary(32)    | Number of bin numbers in REMOTE location: Min 0, Max 6-digit | LC |</p>
<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'881200'</td>
<td>RBYS</td>
<td>9</td>
<td>Binary(8)</td>
<td>Retain by set: 0=NO 1=YES</td>
<td>LV 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'885000'</td>
<td>RLPRI</td>
<td>12</td>
<td>Binary(32)</td>
<td>Required location priority</td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'886000'</td>
<td>RTDJ</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'888800'</td>
<td>RM</td>
<td>9</td>
<td>Binary(8)</td>
<td>Retention method: 0=VRSEL 1=EXPDT</td>
<td>LC OPT, LV VOL, SV(e)</td>
</tr>
<tr>
<td>X'889000'</td>
<td>RMID</td>
<td>25</td>
<td>Character (variable length)</td>
<td>Started procedure name. Up to 17 characters. One of: • procedure name • job name • concatenation of procedure name,identifier</td>
<td>LC</td>
</tr>
<tr>
<td>X'888A00'</td>
<td>RMSB</td>
<td>9</td>
<td>Binary(8)</td>
<td>Retention method set by</td>
<td>LV VOL SV(e)</td>
</tr>
<tr>
<td>X'888A00'</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'88C800'</td>
<td>RTDJ</td>
<td>12</td>
<td>Packed decimal Julian date format</td>
<td>Retention date</td>
<td>LD LV SD SV</td>
</tr>
<tr>
<td>X'88E800'</td>
<td>RTM</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last control data set extract time</td>
<td>LC</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>
</tbody>
</table>

Appendix A. Structured field introducers (SFIs) 91
### Table 16. Command structured field introducers (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'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</td>
<td>LC</td>
</tr>
<tr>
<td>X'894000'</td>
<td>SCI</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 0, Max 9999</td>
<td>LV SV</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>SMI</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>53</td>
<td>Character (variable length)</td>
<td>Server IP address 1-to-45 numeric characters including colon, 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>
<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'8A2800'</td>
<td>STDS</td>
<td>9</td>
<td>Bit (8)</td>
<td>Debug setting</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'8A3200'</td>
<td>STIS</td>
<td>9</td>
<td>Binary (8)</td>
<td>Task - IP verb state</td>
<td>LC</td>
</tr>
</tbody>
</table>

92  
z/OS V2R1.0 DFSMSrmm Application Programming Interface
<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'8A3201'</td>
<td>STIT</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Task - IP verb time</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3203'</td>
<td>STIV</td>
<td>9</td>
<td>Binary (8)</td>
<td>Task - IP verb</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0=NONE 1=READ 2=WRITE 3=CONNECT 4=CLOSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'8A3300'</td>
<td>STLA</td>
<td>10</td>
<td>Binary (15)</td>
<td>Local active tasks Numeric: 0-999</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3307'</td>
<td>STLH</td>
<td>10</td>
<td>Binary (15)</td>
<td>Local held tasks Numeric: 0-999</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3314'</td>
<td>STLO</td>
<td>10</td>
<td>Binary (15)</td>
<td>Local tasks Numeric: 0-999</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3317'</td>
<td>STLR</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Last RESERVE time</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3400'</td>
<td>STNH</td>
<td>9</td>
<td>Binary (8)</td>
<td>New requests held</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0=NOTHELD 1=HELD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'8A3450'</td>
<td>STPL</td>
<td>9</td>
<td>Bit (8)</td>
<td>PDA trace levels</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X'80' level 1 trace X'40' level 2 trace X'20' level 3 trace X'10' level 4 trace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'8A3500'</td>
<td>STQC</td>
<td>12</td>
<td>Binary (32)</td>
<td>Catalog requests Numeric: 0-999999</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3511'</td>
<td>STQN</td>
<td>12</td>
<td>Binary (32)</td>
<td>Nowait requests Numeric: 0-999999</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3515'</td>
<td>STQR</td>
<td>12</td>
<td>Binary (32)</td>
<td>Queued requests Numeric: 0-999999</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3600'</td>
<td>STRF</td>
<td>13</td>
<td>Character (variable length)</td>
<td>Task - requested function</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3602'</td>
<td>STRH</td>
<td>9</td>
<td>Binary (8)</td>
<td>CDS RESERVED 0=DEQ 1=ENQ</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3607'</td>
<td>STRM</td>
<td>9</td>
<td>Binary (8)</td>
<td>RMM status:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0=ACTIVE 1=RESET 2=QUIESCED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'8A3614'</td>
<td>STRT</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Task - requestor's system</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3650'</td>
<td>STSA</td>
<td>10</td>
<td>Binary (15)</td>
<td>Server active tasks Numeric: 0-999</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3657'</td>
<td>STSH</td>
<td>10</td>
<td>Binary (15)</td>
<td>Server held tasks Numeric: 0-999</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3661'</td>
<td>STSL</td>
<td>9</td>
<td>Binary (8)</td>
<td>Server listener task status</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0=Standard or client system 1=task is active 2=task not active</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X'8A3664'</td>
<td>STSO</td>
<td>10</td>
<td>Binary (15)</td>
<td>Server tasks Numeric: 0-999</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3669'</td>
<td>STST</td>
<td>12</td>
<td>Packed decimal time format</td>
<td>Task - Start time</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3700'</td>
<td>STTTQ</td>
<td>16</td>
<td>Character (variable length)</td>
<td>Task - requestor</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3701'</td>
<td>STRTR</td>
<td>11</td>
<td>Character (variable length)</td>
<td>Task - requestor's type: JOB, STC, TSU.</td>
<td>LC</td>
</tr>
<tr>
<td>X'8A3702'</td>
<td>STTSS</td>
<td>9</td>
<td>Binary (8)</td>
<td>Task - status</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0=NONE 1=HOLD 2=CANCEL 3=RESERVE</td>
<td></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'8A3703'</td>
<td>STTT</td>
<td>12</td>
<td>Binary (32)</td>
<td>Task - Token Hexadecimal value: X'00000000' - X'FFFFFFFF'</td>
<td>LC</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 SO</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 LV SD(e) SV(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>hhmmssCC, where hhmmss is the time in hours, minutes, seconds, and tenths of seconds and C is a standard packed-decimal sign character.</td>
<td>LS SS</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'8A7800'</td>
<td>TVXD</td>
<td>12</td>
<td>Binary(32)</td>
<td>TVEXTPURGE days</td>
<td>LC OPT</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</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>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 USE6 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>
</tbody>
</table>
### Table 16. Command structured field introducers (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'8AF001'</td>
<td>VAC</td>
<td>9</td>
<td>Binary(8)</td>
<td></td>
<td>LV SV(e)</td>
</tr>
<tr>
<td>X'8B0000'</td>
<td>VACT</td>
<td>9</td>
<td>Binary(8)</td>
<td></td>
<td>LC</td>
</tr>
<tr>
<td>X'8B0000'</td>
<td>VANX</td>
<td>9</td>
<td>Binary(8)</td>
<td></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></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></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'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 vrrmm</td>
<td>LP LV SP SV(e)</td>
</tr>
<tr>
<td>X'8B4100'</td>
<td>VEX</td>
<td>9</td>
<td>Binary(8)</td>
<td>VRSEL exclude: 0=No 1=Yes</td>
<td>LD SD(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 SO 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>
<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'8BA0000'</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'8BC0000'</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'8BC2000'</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'8BCD0000'</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'8BD0000'</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'8BE0000'</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>
</tbody>
</table>

Appendix A. Structured field introducers (SFIs) 95
<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'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</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>VRSSEL 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</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</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</td>
<td>Last inventory management VRS time</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C4000'</td>
<td>VTYP</td>
<td>9</td>
<td>Binary(8)</td>
<td>Matching VRS type: 0=UNDEFINED</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>EXPDTDROP 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>EXPDTDROP count</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C5D0F'</td>
<td>XDRP</td>
<td>10</td>
<td>Binary(15)</td>
<td>EXPDTDROP percent</td>
<td>LC</td>
</tr>
<tr>
<td>X'8C6000'</td>
<td>XDTJ</td>
<td>12</td>
<td>Packed decimal</td>
<td>Expiration date</td>
<td>LC LD LV SD SV</td>
</tr>
</tbody>
</table>

Table 16. Command structured field introducers (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'8C6100'</td>
<td>XDSB</td>
<td>9</td>
<td>Binary(8)</td>
<td>Expiration date set by</td>
<td>LV VOL, SV(e), LD, SD(e)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0=blank (not set)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1=CMD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2=CMD_DEF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3=CMD_VOLCAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4=OCE_JFCB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5=OCE_EXIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6=OCE_DEF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7=OCE_MAX</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8=OCE_VOLCAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9=LCS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10=LCS_DEF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11=TVEXTPURGE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12=CNVT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13=EXPORT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14=LASTREF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15=OCE_MC</td>
<td></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:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 Exit is not defined or no exit modules exist</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 At least one active exit module exists</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 One or more exit modules exist, but none is active</td>
<td></td>
</tr>
<tr>
<td>X'8C7801'</td>
<td>X200</td>
<td>9</td>
<td>Binary(8)</td>
<td>EDG_EXIT200 installation exit status:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 Exit is not defined or no exit modules exist</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 At least one active exit module exists</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 One or more exit modules exist, but none is active</td>
<td></td>
</tr>
<tr>
<td>X'8C7802'</td>
<td>X300</td>
<td>9</td>
<td>Binary(8)</td>
<td>EDG_EXIT300 installation exit status:</td>
<td>LC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 Exit is not defined or no exit modules exist</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 At least one active exit module exists</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 One or more exit modules exist, but none is active</td>
<td></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>
<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 structured field introducers 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>
<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>ADDDDATASET</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 FRMS MSGL MSGN RCK RSNC RTNC SVCN VOL</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>CHANGETRACK</td>
<td>ENTN MSGL MSGN RSNC RTNC SVCN</td>
</tr>
<tr>
<td>CHANGEVOLUME</td>
<td>CLIB CSG ENTN FRC FRMS MEDN MSGL MSGN RCK RSNC RTNC SVCN VOL</td>
</tr>
<tr>
<td>CHANGEVRS</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 FRMS MSGL MSGN RCK 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 FRMS MSGL MSGN OWN RSNC RTNC SVCN VOL</td>
</tr>
<tr>
<td>LISTBIN</td>
<td>ENTN LCDJ LCID LCSI LCTM LCUD LCUT LINE LOC MIV MOV MEDN MSGL MSGN OVOL RCK RSNC RST 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 CSST CSTM DDT DRR DTE DTM EBIN FBP FCSP FDB FEP FKP FLB FRB FRK FRP FSP FTP FVP FXP IPI JBDT JBTM JDS JRNS JRNU LBN LCT LRK MDS MDT MRG MTR MTRP NOT OPM PACS RBN RC RCF RCT RDMD RTM RUB SAT SDT SID SLR SSOSD SOSP SOST SSM STM UDT UTC UTM VDT VTM XDTJ XTM X100 X200 XX300</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1992, 2013
### Table 17. Structured field introducers by subcommand (continued)

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Structured field introducers</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISTCONTROL LOCDEF</td>
<td>LDAM LDDF LDLC LDLT LDMN LDMT 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 ORIR OROA OROR ORTP ORVE ORVL ORVS</td>
</tr>
<tr>
<td>LISTCONTROL OPTION</td>
<td>ACCT AUD BLP BKPP CATS CDS CMDD CMDO CRP DRP DSDP DSPM DTE EXRB GDGC GDGD IFL JDS JRNF JRTN LCT LCTK LRED MCAT MDS MEDN MOP MRP MSGF MVBY OPM NOT PDAC PDA PDAI PDAL PDAS PSFX PSF2 RC RCF RTBY RUB SID SLM SMP SMUC SMUE SMUS SOSP SRHN SRIP SRPN SRTK SSM SSTD TVXD TVXP UNC VACT VCHG VDRA VDRC VDRP VMIN VREA VREC VREP VRJ VRSL XDRA XDCR XDPR</td>
</tr>
<tr>
<td>LISTCONTROL PARTITION</td>
<td>PTNA PTNL PTSA PTP PTVE 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 SECLEVEL</td>
<td>CLS DNM ERS MSG NME RC SEC SMF</td>
</tr>
<tr>
<td>LISTCONTROL STATUS</td>
<td>STDLS STLS STIT STIV STLA STLH STLO STNH STPL STQC STQN STQR STRF STRH STRM STRT STSA STSH STSL STSO STST STTQ STTR STTS STTT</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 BESK BLKC BLK6 BLKT CDTJ CBJN CLS CPGM CRAT CTLG CTM DC DD DEV DLIRJ DLTD DLWJ DPCT DSEQ DSN DSS6 ENTN FILE LCDJ LCID LCSI LCTM LCUD LCUT LDD LDEV LINE LPGM LRCL LRED LSTP MC MSGL MSGN NME OWIN OXDJ PSZ6 RCFM RSNC RTDJ RTNC SC SG STEP SVCN SYS TZ VEX VJBN VNME VOL VRSR VSCD VSCN VTYP XDSB XDTJ 2JBN 2NME 2SCD 2SCN</td>
</tr>
<tr>
<td>LISTOWNER</td>
<td>ADL DPT EML EMN EMU ENTN ETL FOR ITL LCDJ LCID LCSI LCTM LCUD LCUT LINE MSGL MSGN RSCN RTNC SUR SVCN TZ VLN</td>
</tr>
<tr>
<td>LISTPRODUCT</td>
<td>ENTN FCD LCDJ LCID LCSI LCTM LCUD LCUT LINE MSGL MSGN OWN PDSN PNME PNUM RCK RSNC RTNC SVCN TZ VER VLN VOL</td>
</tr>
<tr>
<td>LISTTRACK</td>
<td>ENTN LCDJ LCID LCSI LCTM LCUD LCUT LINE LOC MEDN MSGL MSGN PID RCK RSNC RST RTNC SVCN VOL</td>
</tr>
<tr>
<td>LISTVOLUME</td>
<td>ACN ACT ADTJ ATM AVL BIN BMN CDTJ CBJN CLS CRAT CRID CTRM CTNR D8N DBMN DEN DESC DEST DLRJ DLWJ DSC DSEQ DSN DSR DSET D12 ENTN EXRB FCD HLD HLOC HLOT INTR KEL1 KEL2 KEM1 KEM2 LBL LCDJ LCID LCSI LCTM LCUD LCUT LDEV LINE LOAN LOC LQFT LVC LYN MDNF MDRX MDTX MEDA MECG MEDN MEDR MEDT MOVM MSGL MSGN MVS NLOC NLOT NME NVL OAC OBMN OBN OCE OLOC OLON OLOT OWN OXDJ PEND PNUM PRD PSZ6 PVL PWT RBY5 RCK RLPD RM RMSB RSNC RTDJ RTNC RTNC SDTJ SEQ SG STVC SVCN TRD TWT TZ UID01 UID02 UID03 UID04 UID05 UID06 UID07 UID08 UID09 UID10 UID11 UID12 USEC USEM USES USE6 VAC VCAP VER VM VMIN VNDR VOL VOLT VOL1 VPCT VRSI VRXI VST VVME WORM WWID XDSB XDTJ</td>
</tr>
<tr>
<td>Subcommand</td>
<td>Structured field introducers</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>LISTVRS</td>
<td>DDTJ DESC DLRJ ENTN LCDJ LCID LCTM LCUD LCUT LINE LOC MSGN NVRS OWN PRTY RET RSNC RTNC SC1 SVCN TLR TYP TZ VANX VDD VJBN VRC VRS VRSI VRXI</td>
</tr>
<tr>
<td>SEARCHBIN</td>
<td>CONT ENTN LINE LOC MEDN MIV MOV MSGN OVOL RCK RSNC RST RTNC SVCN TZ VOL</td>
</tr>
<tr>
<td>SEARCHDATASET</td>
<td>CDTJ CONT CTM DSN ENTN FILE KEYF KEYT LINE LRED MSGN OWN OXDJ RSNC RTDJ RTNC SVCN VOL XDTJ</td>
</tr>
<tr>
<td>SEARCHDATASET(EXPAND= YES)</td>
<td>The same SFIs as the LISTDATASET subcommand.</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= YES)</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 DSR ENTN EXRB HLD HLOC INTR KEYF KEYT LBL LINE LOAN LOC LVC LVN MDNF MDRX MDTX MEDA MEDC MEDN MEDR MDT</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 “EDGXCI: Parameter list”
- The structured field definitions generated by the EDGXSF macro, as shown in “EDGXSF: Structured field definitions”

EDGXCI: Parameter list

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

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

```
MYPL DS 0D ++ EDGXCI PARM LIST
MYPL_XVERSION DS XL1 ++ XVERSION
MYPL_XOPERATION DS XL1 ++ XOPERATION
MYPL_XOPERATION_BEGIN EQU 0 ++ XOPERATION.BEGIN KEYWORD
MYPL_XOPERATION_CONTINUE EQU 1 ++ XOPERATION.CONTINUE KEYWORD
MYPL_XOPERATION_RELEASE EQU 2 ++ XOPERATION.RELEASE KEYWORD
MYPL_XOPERATION_ENDALL EQU 3 ++ XOPERATION.ENDALL KEYWORD
MYPL_XOUTPUT DS XL1 ++ XOUTPUT
MYPL_XOUTPUT_LINES EQU 0 ++ XOUTPUT.LINES KEYWORD
MYPL_XOUTPUT_FIELDS EQU 1 ++ XOUTPUT.FIELDS KEYWORD
MYPL_XEXPAND DS XL1 ++ XEXPAND
MYPL_XEXPAND_YES EQU 0 ++ XEXPAND.YES KEYWORD
MYPL_XEXPAND_NO EQU 1 ++ XEXPAND.NO KEYWORD
MYPL_XAPIADDR DS A ++ XAPIADDR
MYPL_XOUTBUFADDR DS A ++ XOUTBUFADDR
MYPL_XSUBCMDADDR DS A ++ XSUBCMDADDR
MYPL_XTOKEN DS CL4 ++ XTOKEN
MYPL_XMULTI DS XL1 ++ XMULTI
MYPL_XMULTI_NO EQU 0 ++ XMULTI.NO KEYWORD
MYPL_XMULTI_YES EQU 1 ++ XMULTI.YES KEYWORD
MYPL_XRSV0001 DS CL7 ++ RESERVED XRSV0001
MYPL_XRSV0002 DS CL4 ++ RESERVED XRSV0002
MYPL_XRSV0003 DS CL8 ++ RESERVED XRSV0003
MYPL EQU *-MYPL ++ LENGTH OF PLIST
```

Figure 35. Mapping of the parameter list using the list form of EDGXCI

EDGXSF: Structured field definitions

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

- “EDGXSF parameters” on page 104
- “EDGXSF mapping” on page 104
- “EDGXSF labeling conventions” on page 106
**EDGXSF parameters**

The EDGXSF parameters are:

- **DSECT=YES**
  - An optional parameter that specifies whether a DSECT statement is generated. The default is DSECT=YES.
  - **DSECT=YES**
    - Indicates that a DSECT statement should be generated.
  - **DSECT=NO**
    - Indicates that a DSECT statement should not be generated.

- **LIST=YES**
  - An optional parameter that specifies whether the macro expansion is printed. The default is LIST=YES.
  - **LIST=YES**
    - Indicates to print the expansion.
  - **LIST=NO**
    - Indicates do not print the expansion.

- **TITLE=YES**
  - An optional parameter that specifies whether the macro title is printed. The default is TITLE=YES.
  - **TITLE=YES**
    - Indicates to print the title.
  - **TITLE=NO**
    - Indicates do not print the title.

**EDGXSF mapping**

Always use macro EDGXSF to determine the exact labels used to define the DFSMSrmm structured field introducers. The tables in this topic show the dummy control section and the data types that define the generic mapping for the structured field introducers defined in Appendix A, “Structured field introducers (SFIs),” on page 77.

<table>
<thead>
<tr>
<th>Common Name:</th>
<th>API Structure Field Introducers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro ID:</td>
<td>EDGXSF</td>
</tr>
<tr>
<td>DSECT Name:</td>
<td>XSF_SFI</td>
</tr>
<tr>
<td>Owning Component:</td>
<td>DFSMSrmm (DF186)</td>
</tr>
<tr>
<td>Eye-Catcher ID:</td>
<td>None</td>
</tr>
<tr>
<td>Storage Attributes:</td>
<td>Subpool: user specified</td>
</tr>
<tr>
<td>Size:</td>
<td>Variable</td>
</tr>
<tr>
<td>Created by:</td>
<td>Caller</td>
</tr>
<tr>
<td>Pointed to by:</td>
<td>N/A</td>
</tr>
<tr>
<td>Serialization:</td>
<td>None</td>
</tr>
<tr>
<td>Function:</td>
<td>The XSF_SFI area is initialized by DFSMSrmm when an API call is made via the EDGXCI executable macro</td>
</tr>
</tbody>
</table>
### Table 18. Structure XSF_OUTBUF

<table>
<thead>
<tr>
<th>Offset</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</td>
<td>0</td>
<td>(0)</td>
<td>STRUCTURE</td>
<td>*</td>
<td>XSF_OUTBUF</td>
<td>Output buffer</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>(0)</td>
<td>SIGNED</td>
<td>4</td>
<td>XSF_OUTBUF_BUFLNG</td>
<td>Output buffer length</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>(4)</td>
<td>SIGNED</td>
<td>4</td>
<td>XSF_OUTBUF_RQDLNG</td>
<td>Required buffer length</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>(8)</td>
<td>SIGNED</td>
<td>4</td>
<td>XSF_OUTBUF_DATALNG</td>
<td>Length of output data</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>(C)</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>Offset</th>
<th>Dec</th>
<th>Hex</th>
<th>Type</th>
<th>Len</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>(0)</td>
<td>STRUCTURE</td>
<td>*</td>
<td>XSF_SFI</td>
<td>Structured field introducers</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>(0)</td>
<td>CHARACTER</td>
<td>8</td>
<td>XSF_SFI_HD</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>(0)</td>
<td>SIGNED</td>
<td>2</td>
<td>XSF_SFI_LENGTH</td>
<td>Length</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>(2)</td>
<td>CHARACTER</td>
<td>3</td>
<td>XSF_SFI_ID</td>
<td>Identifier</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>(2)</td>
<td>CHARACTER</td>
<td>2</td>
<td>XSF_SFI_IDVAL</td>
<td>Identifier value</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>(4)</td>
<td>CHARACTER</td>
<td>1</td>
<td>XSF_SFI_IDQUAL</td>
<td>Identifier qualifier</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>(5)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_TYPE</td>
<td>Type</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>(7)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_DTYPE</td>
<td>Data type</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>(8)</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>Offset</th>
<th>Dec</th>
<th>Hex</th>
<th>Type</th>
<th>Len</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>(8)</td>
<td>STRUCTURE</td>
<td>14</td>
<td>XSF_SFI_COMPTYPE1</td>
<td>Compound section</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>(8)</td>
<td>CHARACTER</td>
<td>6</td>
<td>XSF_SFI_COMPDATA</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>(8)</td>
<td>CHARACTER</td>
<td>6</td>
<td>XSF_SFI_COMPHDR</td>
<td>Compound header</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>(8)</td>
<td>CHARACTER</td>
<td>6</td>
<td>XSF_SFI_COMPENT</td>
<td>Compound entry</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>(8)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_COMPTYPE</td>
<td>Compound type</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>(9)</td>
<td>CHARACTER</td>
<td>3</td>
<td>XSF_SFI_FIELD1</td>
<td>Length of first field</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>(9)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_LEN1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>(A)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_DTYP1</td>
<td>Type of first field</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>(B)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_FACTOR</td>
<td>Factor for second field</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>(C)</td>
<td>CHARACTER</td>
<td>2</td>
<td>XSF_SFI_FIELD2</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>(C)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_LEN2</td>
<td>Length of second field</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>(D)</td>
<td>UNSIGNED</td>
<td>1</td>
<td>XSF_SFI_DTYP2</td>
<td>Type of second field</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>(E)</td>
<td>CHARACTER</td>
<td>8</td>
<td>XSF_SFI_COMPVAL</td>
<td>The value</td>
</tr>
</tbody>
</table>

**Data Types** (XSF_SFI_DTYPE, XSF_SFI_DTYP1, XSF_SFI_DTYP2)

<table>
<thead>
<tr>
<th>Len</th>
<th>Type</th>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HEX</td>
<td>00</td>
<td>XSF_SFI_DTYPE_UNDEF</td>
<td>Undefined data</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>01</td>
<td>XSF_SFI_DTYPE_CHAR_FIX</td>
<td>N byte character</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>02</td>
<td>XSF_SFI_DTYPE_BITFLAG</td>
<td>Bit flag byte (8 bits)</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>03</td>
<td>XSF_SFI_DTYPE_BIN8</td>
<td>1 byte (hex) value</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>04</td>
<td>XSF_SFI_DTYPE_BIN15</td>
<td>2 byte hex value</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>05</td>
<td>XSF_SFI_DTYPE_BIN31</td>
<td>4 byte hex value</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>06</td>
<td>XSF_SFI_DTYPE_BIN64</td>
<td>8 byte hex value</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>07</td>
<td>XSF_SFI_DTYPE_CHAR_VAR</td>
<td>Variable length character</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>08</td>
<td>XSF_SFI_DTYPE_COMPOUND</td>
<td>Compound SFI</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>09</td>
<td>XSF_SFI_DTYPE_UDATE</td>
<td>4 byte packed decimal date YYYYDDD</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>0A</td>
<td>XSF_SFI_DTYPE_TIME</td>
<td>4 byte packed decimal time HHMMSSST</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>1</td>
<td>HEX</td>
<td>00</td>
<td>XSF_SFI_COMPTYPE_UNDEF</td>
<td>Undefined type</td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>01</td>
<td>XSF_SFI_COMPTYPE_FACTOR</td>
<td>Factored type</td>
</tr>
</tbody>
</table>

**Factors** (XSF_SFI_Factor)
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

Here is a sample mapping of the Begin and End ACCESS group:

<table>
<thead>
<tr>
<th>Len</th>
<th>Type</th>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>HEX</td>
<td>0008021000000000</td>
<td>XSF_SFI_ACCESS</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>HEX</td>
<td>021000</td>
<td>XSF_SFI_ID_ACCESS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HEX</td>
<td>0008</td>
<td>XSF_ACCESS_LENGTH</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>HEX</td>
<td>0008021008000000</td>
<td>XSF_SFI_EACCESS</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>HEX</td>
<td>021080</td>
<td>XSF_SFI_ID_EACCESS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HEX</td>
<td>0008</td>
<td>XSF_EACCESS_LENGTH</td>
<td></td>
</tr>
</tbody>
</table>

The VOL and VRS groups are defined using this format:
- XSF_SFI_ID_xxx and XSF_xxxGRP_LENGTH
- XSF_SFI_ID_Exxx and XSF_ExxxGRP_LENGTH

Here is a sample mapping of the Begin and End VOL group:

<table>
<thead>
<tr>
<th>Len</th>
<th>Type</th>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>HEX</td>
<td>0008036000000000</td>
<td>XSF_SFI_VOLGRP</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>HEX</td>
<td>036000</td>
<td>XSF_SFI_ID_VOL</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HEX</td>
<td>0008</td>
<td>XSF_VOLGRP_LENGTH</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>HEX</td>
<td>0008036008000000</td>
<td>XSF_SFI_EVOLGRP</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>HEX</td>
<td>036080</td>
<td>XSF_SFI_ID_EVOL</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HEX</td>
<td>0008</td>
<td>XSF_EVOLGRP_LENGTH</td>
<td></td>
</tr>
</tbody>
</table>

Labeling: Structured field introducers that introduce data

Structured field introducers introduce data and are defined using this format:
- XSF_SFI_xxxx_ID
- XSF_xxxx_LENGTH
- XSF_xxxx_DTYPE

Here is a sample mapping of the ATM SFI:
### Labeling: Flags
Output data for some structured field introducers are defined as bit flags using this format: `XSF xxxx_FLAG_name`.

Here is a sample mapping of the ACT SFI:

<table>
<thead>
<tr>
<th>Len</th>
<th>Type</th>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>HEX</td>
<td>0009802000000002</td>
<td>XSF_SFI_ACT</td>
<td>Actions on release</td>
</tr>
<tr>
<td>3</td>
<td>HEX</td>
<td>802000</td>
<td>XSF_SFI_ACT_ID</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HEX</td>
<td>0009</td>
<td>XSF_ACT_LENGTH</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>02</td>
<td>XSF_ACT_DTYPE</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>80</td>
<td>XSF_ACT_FLAG_SCRATCH</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>40</td>
<td>XSF_ACT_FLAG_REPLACE</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>20</td>
<td>XSF_ACT_FLAG_INIT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>10</td>
<td>XSF_ACT_FLAG_ERASE</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>08</td>
<td>XSF_ACT_FLAG_RETURN</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>04</td>
<td>XSF_ACT_FLAG_NOTIFY</td>
<td></td>
</tr>
</tbody>
</table>

### Labeling: Bin(8) data
Output data for some structured field introducers are defined as one-byte binary numbers using this format: `XSF xxxx_DATA_name`.

Here is a sample mapping of the LOCT SFI:

<table>
<thead>
<tr>
<th>Len</th>
<th>Type</th>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>HEX</td>
<td>000984E000000003</td>
<td>XSF_SFI_LOCT</td>
<td>Location type</td>
</tr>
<tr>
<td>3</td>
<td>HEX</td>
<td>84E000</td>
<td>XSF_SFI_LOCT_ID</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HEX</td>
<td>0009</td>
<td>XSF_LOCT_LENGTH</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HEX</td>
<td>03</td>
<td>XSF_LOCT_DTYPE</td>
<td></td>
</tr>
<tr>
<td>NUMB</td>
<td>HEX</td>
<td>00</td>
<td>XSF_LOCT_DATA_SHELF</td>
<td></td>
</tr>
<tr>
<td>NUMB</td>
<td>HEX</td>
<td>01</td>
<td>XSF_LOCT_DATA_STORE_BUILTIN_BINS</td>
<td></td>
</tr>
<tr>
<td>NUMB</td>
<td>HEX</td>
<td>02</td>
<td>XSF_LOCT_DATA_MANUAL</td>
<td></td>
</tr>
<tr>
<td>NUMB</td>
<td>HEX</td>
<td>03</td>
<td>XSF_LOCT_DATA_AUTO</td>
<td></td>
</tr>
<tr>
<td>NUMB</td>
<td>HEX</td>
<td>04</td>
<td>XSF_LOCT_DATA_STORE_BINS</td>
<td></td>
</tr>
<tr>
<td>NUMB</td>
<td>HEX</td>
<td>05</td>
<td>XSF_LOCT_DATA_STORE_NOBINS</td>
<td></td>
</tr>
<tr>
<td>NUMB</td>
<td>HEX</td>
<td>06</td>
<td>XSF_LOCT_DATA_INCTR</td>
<td></td>
</tr>
</tbody>
</table>

### 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 36] 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 51. 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 36] shows:

0000 0000100000000000 0000084000008260 0000000001A82A0 0000007D9D04E4
0020 E2C5D948C69C5D3 C4BE3C5E2E300E 88C000000001E566 D3F0F0F1000F8700
0040 000000079D404E4 E2C5D9000C8A9E00 000000FFF9D9000 0CB13000000000920
0060 05320F000C81A000 000000658266F00 0C833000000005500 00000100002680
0080 0000000000000000 0000000000000000 0000000000000000 0000000000000000
00A0 0FFC 0000000000000000 0000000000000000 0000000000000000 0000000000000000
00C0 0FFE 0000000000000000 0000000000000000 0000000000000000 0000000000000000

[Figure 36. Hexadecimal representation of the contents of an output buffer]

The first and second lines of the output buffer shown in [Figure 36] show:

© Copyright IBM Corp. 1992, 2013
The second and third lines of the output buffer shown in Figure 36 on page 109 show:

```
0020 D906D5C54BC6C9C5 D3C44BE3C5E2E300 0E8BC000000001E5 D6D3F0F0F1001087
0040 0000000007D6E6D5C5D9D6D5C5 300000009199711 7C000C81A0000000
```

• **Owner structured field**
  - 0010870000000007 D6E6D5C5D9D6D5C5

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

• **Create Date structured field**
  - 000C813000000009 1997117C

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

The third and fourth lines of the output buffer shown in Figure 36 on page 109 show:

```
0040 0000000007D6E6D5 C5D9D6D5C5000C81 000000009199711 7C000C81A0000000
0060 0A0815270C000C83 3000000005000000 0100000260000000 0000000000000000
```

• **Create Time structured field**
  - 000C81A000000000A 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 37 on the address of the output buffer you are interested in.

2. Base the XSF_SFI definition in macro EDGXSF as shown in Figure 38 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 (SFIs)," on page 77, 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_DATALENGTH.

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

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

8. Move to the end of the SFI by adjusting the pointer. In this example, when your pointer is at offset x'00000071' into the output buffer, there are two indicators that you are done with the contents of the buffer:
   - You are looking at the End DATASET group SFI.
Note: This is true only if you did not specify MULTI=YES in your call to the API. If you use MULTI=YES, your output buffer may contain more than one resource group.

- Adjusting the XSF_SFI pointer by the length of this SFI (8 bytes) points you past the last byte of data in the buffer.

9. Repeat these steps to process each structured field.

In the examples shown in Figure 37 on page 111 and Figure 38 on page 111:

- Adding the length of the data (‘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 structured field introducers 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

Publications for this product are offered in Adobe Portable Document Format (PDF) and XHTML through the z/OS Information Center at http://publib.boulder.ibm.com/infocenter/zos/v2r1/index.jsp. If you experience difficulty with the accessibility of any z/OS information, send an email to mhvrdfs@us.ibm.com or write to:

IBM® Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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.

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.
The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3 * * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next
higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

• * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Note:
1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

• + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.
Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not 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 character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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.

**Minimum supported hardware**

The minimum supported hardware for z/OS releases identified in z/OS
announcements can subsequently change when service for particular servers or
devices is withdrawn. Likewise, the levels of other software products supported on
a particular release of z/OS are subject to the service support lifecycle of those
products. Therefore, z/OS and its product publications (for example, panels,
samples, messages, and product documentation) can include references to
hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for
  z/OS](http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM
  representative.

**Programming interface information**

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

**Trademarks**

DFSMSrmm
IBM
IBMLink
RACF
z/OS
z/VM

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

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

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

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

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

Numerics
2JBN ('8C8000') - Secondary VRS Jobname Mask 97
2NME ('8C9000') - Secondary VRS Mask 97
2SCD ('8CA000') - Secondary VRS Subchain Start Date 97
2SCN ('8CB000') - Secondary VRS Subchain Name 97

A
abbreviations for subcommands 2
ABND ('800800') - closed by Abend 81
accessibility 113
contact IBM 113
features 113
account number SFI 81
accounting source SFI 81
ACCT ('800800') - Accounting Source 81
ACN ('801000') - Account Number 81
ACS ('801800') - SMSACS 81
ACT ('802000') - Actions on release 81
Action Status SFI 81
actions on release SFI 81
Actions Pending SFI 90
ADDBIN
SFIs for 59
subcommand abbreviation 2
ADDDATASET
SFIs for 59
subcommand abbreviation 2
ADDDOWNER
SFIs for 59
subcommand abbreviation 2
ADDPROMOCT
SFIs for 59
subcommand abbreviation 2
ADDRACK
SFIs for 59
subcommand abbreviation 2
address line SFI 81
ADDVOLUME
SFIs for 59
subcommand abbreviation 2
ADDVRS
SFIs for 59
subcommand abbreviation 2
ADR ('803001') - Address Line 81
ADTJ ('804000') - Assigned Date 81
API methods 20
application programming interface
mapping macros 103
assigned date SFI 81
assigned time SFI 81
assistive technologies 113
AST ('805000') - Action Status 81
ATM ('806000') - Assigned Time 81
AUD ('807000') - SMF Audit Record Number 81
authentication 28
Automove SFI 86
AVL ('808000') - Volume Availability 81
backup procedure name SFI 81
BDTJ ('809000') - Last Control Data Set Backup Date 81
Begin and End Resource group structured field introducers 78
Begin and End Resource groups 56
BESK ('809310') - CA Tape Encryption key index 81
BIN ('80A000') - Bin Number 81
Bin Count SFI 82
Bin Number Media Name SFI 81
bin number SFI 81
Bin Numbers in DISTANT SFI 83
Bin Numbers in LOCAL SFI 85
Bin Status SFI 91
Bin(8) data labeling 107
BKPP ('80B000') - Backup Procedure Name 81
BLK6 ('80D0B0') - Total block count 81
BLKC ('80C000') - Block Count 81
BLKS ('80D000') - Block Size 81
BLKT ('80D030') - Total block count 81
block count SFI 81
block size SFI 81
BLP ('80E000') - BLP Option 81
BLP option SFI 81
BMN ('80F000') - Bin Number Media Name 81
BTM ('810000') - Last Control Data Set Backup Time 81
C++ classes 20
CA Tape Encryption key index 81
CAct ('811000') - Control Active Functions 82
Catalog requests SFI 93
Catalog status SFI 82
Catalog Synchronize Date 82
Catalog Synchronize in Progress SFI 84
Catalog Synchronize Time 82
CATRETPD Retention Period SFI 82
CATS ('811800') - CATSYSID Value 82
CATSYSID Value 82
CDS ('812000') - Control Data Set Identifier 82
CDS RESERVEd SFI 93
CDSQ ('812900') - Control Data Set ENQ 82
CDSU ('812100') - Control Data Set Percentage Used SFI 82
CDTJ ('813000') - Create Date 82
CHANGEDATASET
SFIs for 59
subcommand abbreviation 2
CHANGEOWNER
SFIs for 59
subcommand abbreviation 2
CHANGEPRODUCT
SFIs for 59
subcommand abbreviation 2
CHANGEVOLUME
SFIs for 59
subcommand abbreviation 2
character set chart xii
use in statement xii
CJBN ('814000') - Job Name 82
CLIB ('815000') - Current Library Name 82
Client IP address SFI 82
Client/server host name SFI 82
closed by Abend SFI 81
CLS ('816000') - Security Class Description 82
CMDD ('816900') - Command Authorization - DSN SFI 82
CMDO ('8169A0') - Command Authorization - Owner SFI 82
CNT ('817000') - Bin, Rack, or Volume Count 82
Command Authorization - DSN SFI 82
Command Authorization - Owner SFI 82
command classes 20
Common Time SFI 94
compound SFI 77
Compression ratio in hundreths SFI 82
CONT ('057000') - Continue 80
CONT SFI 80
CONTINUE Operation 29
continuing a request 7, 73
Control Active Functions SFI 82
Control Data Set Create Date SFI 87
Control Data Set Create Time SFI 88
Control Data Set ENQ SFI 82
Control Data Set Identifier SFI 82
Control Data Set Name SFI 87
Control Data Set Percentage Used SFI 82
Control Data Set Type SFI 88
count of volumes stacked on a stacked volume SFI 94
CPGM ('817820') - Creating program name 82
CRAT ('817890') - Compression ratio in hundreths 82
Create Date SFI 82
Create Time SFI 82
Creating program name SFI 82
Creating system ID for first file SFI 84
CRID ('817900') - File 1 create user ID 82
DELETEVOLUME (continued)
subcommand abbreviation 2
DELETEVRS
SFIs for 60
subcommand abbreviation 2
delimiters xii
DEN (81F000) - Media Density 83
DESC (820000') - Volume or VRS Description 83
DEST (821000) - Destination Name 83
Destination bin name SFI 83
Destination bin number SFI 82
Destination Name SFI 83
Destination Type SFI 83
DEV (822000') - Device Number 83
Device Number SFI 83
DFSMSrmm API 15
DFSMSrmm API Command C++
Classes 20
DFSMSrmm API Command Classes 20
DFSMSrmm API Command Java
Classes 20, 21
DFSMSrmm API with Web services 25
DFSMSrmm System ID SFI 92
Disposition DD name SFI 83
Disposition Message Prefix SFI 83
DLR/DIRJ (823000') - Date Last Referenced/Read 83
DLTD (823700) - Deleted by disposition processing 83
DLWJ (824000) - Date Last Written 83
DNM (825000) - Data Set Name Mask 83
DPCT (825E00') - Percent of volume 83
DPT (826000') - Owner's department 83
DRP (827000) - Default Retention Period 83
DSC (828000') - Data Set Count 83
DSEQ (829000') - Data Set Sequence 83
DSN (82A000') - Data Set Name 83
DSPD (82A500') - Disposition DD name 83
DSPM (82AA00') - Disposition message prefix 83
DSR (82B000') - Data Set Recording 83
DSS6 (82B030) - Data Set Size 83
DSTT (82B200) - Destination Type 83
DSYS (82BB00') - Creating system ID for first file 84
DTE (82C000) - Installation Date Format 84
DTM (82D000) - Last Store Update Run Time 84

DELETEVOLUME
SFIs for 60
subcommand abbreviation 2
Deleted by disposition processing SFI 83
DELETEDATASET
SFIs for 60
subcommand abbreviation 2
DELETEOWNER
SFIs for 60
subcommand abbreviation 2
DELETERPRODUCT
SFIs for 60
subcommand abbreviation 2
DELETERACK
SFIs for 60
subcommand abbreviation 2
DELETEVOLUME
SFIs for 60

EDGXC1: Call DFSMSrmm Interface 3
EDGXCI
reason codes 9
return codes 9
EDGXCI macro
specifying TSO subcommand input in 29
EDGXCI macro syntax 5
EDGXSF
labeling conventions 106
mapping 104
parameters 104
EDGXSF Structured Field
Definitions 103
EDGXSF Structured Field
Free Bins in DISTANT Location 84
field format for data 48
FILE (833000') - Physical File Sequence 84
File 1 create user ID 82
flags
labeling 107
FLB (834000') - Free Bin Numbers in LOCAL 84
FORE (835000') - Owner's Forename 84
FRB (836000') - Free Bin Numbers in REMOTE 84
FRC (400000') - Function Return Code 97
FRC (400000') - Function Return Code 79
Free Bin Numbers in LOCAL SFI 84
Free Bin Numbers in REMOTE SFI 84
Free Bins in DISTANT Location SFI 84
Free Rack Numbers in Library SFI 84
freeing resources 38
FRK (837000') - Free Rack Numbers in Library 84
FRS (401000') - Function Reason Code 79
Function Reason Code SFI 79
Function Return Code SFI 79
null
M
management class attributes enabling SFI 87
Management Class SFI 87
mapping macros
EDGXXCI 103
EDGXXSF 103
Master Overwrite SFI 88
Matching VRS Job Name SFI 95
Matching VRS Name SFI 95
Matching VRS Type SFI 96
MAXHOLD Value SFI 92
Maximum Retention Period SFI 88
MC (851000) - Management Class 87
MCAT (851200) - management class attributes enabling 87
MDNF (851400) - Media Information Name 87
MDRA (851980) - MEDINF replace policy 87
MDRP (8519C0) - MEDINF replace policy 87
MDRT (8519E0) - MEDINF replace policy 87
MDRW (8519F0) - MEDINF replace policy 87
MDRX (851A00) - External Recording Technology 87
MDS (852000) - Control Data Set Name 87
MDTJ (853000) - Control Data Set Create Date 87
MDTX (853400) - External Media Type 87
MEDA (854000) - Media Special Attributes 87
MEDC (855000) - Media Compaction 87
Media Compaction SFI 87
Media Density SFI 83
Media Information Name SFI 87
Media Name SFI 87
Media Recording Format SFI 87
Media Special Attributes SFI 87
Media Type SFI 88
MEDN (856000) - Media Name 87
MEDR (857000) - Media Recording Format 87
MEDT (858000) - Media Type 88
memory size limitation 26
Message Line SFI 80
Message Number SFI 80
Message SFIs 80
Message Text Case SFI 88
Message Variable SFIs 80
message variables
structured field introducers 57
messages
structured field introducers 57
MSGF (85E000) - Case of Message Text 88
MSGL (051000) - Message Line 80
MSGN (052000) - Message Number 80
MST (85F000) - Move Status 88
MTM (860000) - Control Data Set Create Time 88
MTO (861000) - Target Location Name 88
MTP (862000) - Control Data Set Type 88
MTY (862800) - Move Type 88
multiple parameter list, multiple token areas 37
multiple parameter list, single token area 36
MVBY (86B200) - Move By 88
MVS (863000) - MVS Use 88
MVS Use SFI 88
N
navigation
keyboard 113
New requests held SFI 93
Next Vital Record Specification Name SFI 88
Next Volume SFI 88
Next VRS Value SFI 95
NLOC (865000) - Required Location 88
NLOT (865200) - Required location type 88
NME (866000) - Security Class Name 88
NOSMT action for partition entry SFI 90
NOT (866800) - Notify 88
Notices 117
Nowait requests SFI 93
Number of Bin Numbers in REMOTE SFI 90
Number of Entries SFI 80
Number of Volumes SFI 95
NVL (867000) - Next Volume 88
NVRS (868000) - Next VRS Name 88
O
OAC (869000) - Owner Access 89
OBMN (86A000) - Old Bin Number Media Name 89
OBN (86B000) - Old Bin Number 89
obtaining space for output buffer 12
OCE (86B800) - Volume Information Recorded at O/C/EOV 89
Offset to Message ID SFI 92
Old Bin Number Media Name SFI 89
Old Bin Number SFI 89
Old loan location SFI 89
Old Location SFI 89
Old location type SFI 89
Old volume SFI 89
OLOC (86C000) - Old Location 89
OLOM (86C100) - Old loan location type 89
OLOT (86C200) - Old location type 89
Operating Mode SFI 89
OPL (86D000) - Position of Rack Number or Pool ID 89
OPM (86E000) - Operating Mode 89
ORIA (86E8A0) - Input action 89
Original Expiration Date SFI 89
ORIR (86E8A8) - Input ignore condition 89
ORIV (86E8A8) - Input reject condition 89
OROA (86E8A0) - Output action 89
OROI (86E8A8) - Output ignore condition 89
OROR (86E8A8) - Output reject condition 89
ORTP (86E8F8) - Type of open rule entry 89
ORVE (86EF8) - Volume range end 89
ORVL (86EF85) - Volume serial number 89
ORVS (86EF80) - Volume range start 89
Output action SFI 89
output buffer
hexadecimal example of an output buffer 109
obtaining space for 12
processing contents of 111
Output Data Line SFI 86
Output ignore condition SFI 89
Output reject condition SFI 89
OVL (86F000) - Position of Volume Serial 89
OVR (86F500) - Old volume 89
OWN (870000) - Owner 89
Owner Access SFI 89
Product Number: 5650-ZOS

Printed in USA